Quality of Experience for the Operation of a Small Scale Ground Vehicle over Unreliable Links

A thesis presented to

the faculty of

the Scripps College of Communication of Ohio University

In partial fulfillment

of the requirements for the degree

Master of Science

Abdoulaye Saadou Yaye

August 2015

This thesis titled

Quality of Experience for the Operation of a Small Scale Ground Vehicle over Unreliable Links

by

ABDOULAYE SAADOU YAYE

has been approved for

the School of Information and Telecommunication Systems

and the Scripps College of Communication by

Julio Arauz

Assistant Professor of Information & Telecommunication Systems

Scott Titsworth

Dean, Scripps College of Communication

Abstract

SAADOU YAYE, ABDOULAYE., M.S, August 2015, Information and Telecommunication Systems

Quality of Experience for the Operation of a Small Scale Ground Vehicle over Unreliable Wireless Links

Director of Thesis: Julio Arauz

This thesis proposes a compensation mechanism for maintaining the quality of experience (QoE) when a user controls a ground vehicle over a wireless link affected by packet losses. This research is important because it applies to many relevant areas such as remote surgery, and search and rescue operations in human unfriendly environments. Previous research focused on quantifying network parameters, and using a delay tolerant network (DTN) approach to compensate for packet losses. This study proposes a different approach by using a heuristic based algorithm that tracks packet losses and then decides a course of action based on the current state of the system.

Such a strategy helps reduce the mean time to complete tasks (TTCT) by approximately 16, 29, 36 and 90% when the link condition is in the good, medium, bad, and worst condition respectively. The system is successfully tested via a human-based experimental study by approximately 60 participants.

Table of Contents

Page

Abstract . . . iii

List of Tables . . . v

List of Figures . . . vi

Chapter 1: Introduction . . . 1
Motivation . . . 2

Chapter 2: Literature Review . . . 3
2.1 Related Work . . . 3
2.2 Research Differentiation . . . 15
2.3 Background on Methods Proposed to Test the Hypothesis . . . 15

Chapter 3: System and Experimental Design . . . 19
3.1 Research Description . . . 19

Chapter 4: Discussion of Results . . . 42
4.1 Statistical Analyses . . . 42

Chapter 5: Conclusions . . . 59
5.1 Summary . . . 59
5.2 Limitations . . . 59

References . . . 60

A: Link Emulator . . . 62

B: Description of the Hardware and the Mode of Operation . . . 64

C: Description of the Software . . . 71

1. SECOND APPENDIX WITH A LONGER TITLE - MUCH LONGER IN FACT 135

1.1 Appendix Section . . . 135

List of Tables

Table		Page
1	Mean duration (in millisecond) of the up state (i.e. $1/\alpha$) and the down state (i.e. $1/\lambda$) used for each link condition	23
2	Up state duration (in milliseconds) statistics for different distances between the transmitter and the receiver	25
3	Down state duration (in milliseconds) statistics for different distances between the transmitter and the receiver	25
4	Variables name and operation	40
5	Variables name and type	41
6	t-Test in the very good link conditions	44
7	t-Test in the good link conditions	46
8	t-Test in the medium link conditions	48
9	t-Test in the bad link conditions	50
10	t-Test in the worst link conditions	52
11	Different sets of mappings between φ and ε_e.	54
12	Percentage (%) decrease in TTCT due to compensation with respect to tasks and link conditions.	56
13	Statistical summary for the good state	62
14	Statistical summary for the bad state	63
15	SPI pin arrangement slave A board-transceiver A	66

List of Figures

Figure		Page
1	Test best for quantifying the quality of an actuator	4
2	Test scenarios for evaluating the response of a robotic arm	5
3	Test best for the experimental approach	7
4	Collection infrastructure	10
5	Test bed for the METERON experiment	11
6	Testing the OPSCOM-1	14
7	Experimental testbed for testing QoE compensation mechanism	20
8	Task trajectories for the CPD	21
9	Two-state transition diagram	22
10	Up state at 15 meters	26
11	Down state at 15 meters	27
12	Up state at 20 meters	28
13	Down state at 20 meters	29
14	Up state at 25 meters	30
15	Down state at 25 meters	31
16	Mapping function, $\varepsilon_e(\varphi)$	33
17	Example of the evolution of φ with time	34
18	State transition diagram. φ represents the state of the system and varies when a packet is received	35
19	Mode of operation of ϱ	37
20	Using ϱ to determine the size of runs of packets in-between delay	38
21	Mapping function, $\varepsilon_e(\varphi, \varrho)$	39
22	Mean TTCT in the very good link condition with a confidence level of 95%.	45

23 Mean TTCT in the good link condition with a confidence level of 95%. . . 47
24 Mean TTCT in the medium link condition with a confidence level of 95%. . 49
25 Mean TTCT in bad link condition with a confidence level of 95%. 51
26 Mean TTCT in the worst link condition with a confidence level of 95%. . . 53
27 Mean TTCT in good channel condition with a confidence level of 95%. 55
28 Mean TTCT in bad channel condition with a confidence level of 95%. 55
29 Trend of percentage decrease in TTCT 57
30 Schematic of the joystick 65
31 Circuit diagram of the controller 67
32 Circuit diagram of the CPD 68
33 Possible types of motions for the CPD 70
1.1 TAMU figure 135

Chapter 1: Introduction

This project will examine how to compensate for wireless link quality variations when a ground vehicle is commanded by a human operator over a loss prone packet-switched link. It aims at quantifying how to maintain quality of experience (QoE) under these circumstances. In this work, QoE is defined as a quantifiable variable that is a function of the network characteristics and is represented by the time to complete predetermined tasks (TTCT). In general QoE refers to the level of satisfaction from a user's point of view with respect to the performance of a system. The system employed in this study consists of a cyber-physical device (CPD) which is a small scale ground vehicle whose frame is supported by four direct current (DC) motors arranged symmetrically allowing specific types of motion over the ground surface. This work aims at compensating for packet losses that result in QoE degradation under unfavorable link conditions.

Compensation means replicating packets when loss occurs during packet transit over the wireless link between the user and the CPD. Notice that the wireless link does not provide a reliable data transmission (RDT) because it is a real-time application. The system does not benefit from a forward error correction (FEC). A cyclic redundancy check (CRC) enables the receiver to detect packet error and discard the affected packets. However, re-transmission of lost or corrupt packets does not take place because it is assumed that packet acknowledgment is not necessary since the packet could arrive too late when re-transmitted.

This research is carried out on the basis of two hypothesizes. A null hypothesis asserts that no compensation can be achieved for packet losses, while an alternative hypothesis supports that it should be possible to compensate for packet losses. The outcome of this study will answer the question with regards to which hypothesis should be maintained or rejected.

This document starts with a discussion on the importance of the research topic. Then a review on existing related studies is presented in Chapter 2, focusing on previous work on the quality of operation of actuators under lossy conditions.

This is relevant because it provides an insight on how the qualitative and quantitative performance of an actuator relate to network performance metrics such as packet loss and variation of delay. Next, another work titled "Quality of Experience from the User and Network Perspective" is explored. The latter highlights the dependence of the user's perception of quality on some network metrics (i.e. packet loss and data throughput), and how a change in one parameter could affect another. Finally, the literature review also includes a discussion with regards to how the National Aeronautics and Space Administration (NASA) collaborated with the European Space Agency (ESA) in order to promote exploration of planets by humans in such a way that they can remotely control robots from orbiting stations such as the International Space Station (ISS).

Chapter 3 presents the research questions and an explanation of the uniqueness of the study. In addition, the scope is stated and a framework is developed. Chapter 3 also explains how the measurements were performed and how evaluation of QoE took place as a function of packet loss. The conclusion and the discussion of results are presented in Chapter 4. Finally, the Appendix includes details on both the hardware and software components used in this research.

1.1 Motivation

Previous work on compensation mechanisms for the operation of a CPD over unreliable wireless links is not extensive. This project is relevant as the perception of QoE matters when a user tries to operate a remote system in spite of the lack of constant guaranteed access to a reliable communications channel. It could be useful in multiple areas. For example, it could apply to exploratory vehicles in human unfriendly sites. Such sites could be environments with buried mines where search and rescue operations are of a great necessity. Health facilities where remote surgery is needed; planet exploration are also examples of areas where this research could be useful [5] [12]. Moreover, submarines and other systems (e.g actuators and flying vehicles) could also benefit from this research, especially with the growing need for exploring locations such as deep oceans and many other hostile places (e.g. bio-hazard sites).

Chapter 2: Literature Review

2.1 Related Work

Earlier related studies were conducted in different domains. In a first study, techniques used to quantify QoE over an actuator platform were analyzed under various levels of degraded network metrics (variation of packet delay and packet loss) [5]. A second study considered employed two different strategies to predict QoE. The first was based on measurements conducted over an experimental platform, while the second one was non-experimental, involving only sessions of data collection and analysis using an existing large scale real-time user network [6]. This section also includes a discussion of the multi-purpose end-to-end robotic operations network (METERON) project which looks at how a robust communication protocol was gradually set up to avoid loss of information during communication between a robot located on the ground and a remote operator.

2.1.1 Quality of Experience for the Operation of Actuators Under Lossy Conditions

Recent research was conducted with the aim of studying QoE when a packet based platform is used to control a robot arm over a wired-network using a set of changing factors. The work focused on quantifying how a human operator would perceive the performance of an actuator in the presence of packet delay variation and packet losses [5]. The test bed used for the study, where elements such as a joystick, an primary electronic board, a link emulator, a secondary electronic board and a robotic arm are respectively cascaded, as shown in Fig. 1. The description of the elements shown in the figure follows.

- Item (a) represents a joystick. It generates the analog signals when activate by a user.
- Item (b) is a first electronic board used to convert commands from the controller into network layer packets.
- Device (c) represents a link emulator used to delay or discard packets.
- Item (d) is an interfacing electronic board between the emulator and the robot arm.
- Item (e) is a robotic arm composed of different actuators.

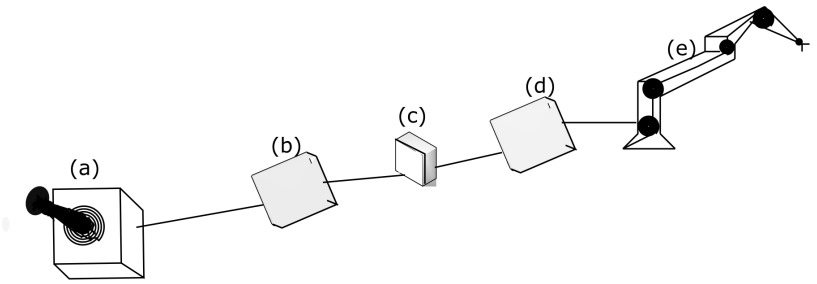

Figure 1: Test best for quantifying the quality of an actuator

The goal of this platform was to quantify network QoE under unfavorable network metrics. This work intended to predict the effect on QoE of changes in the percentage of packets lost and in packet delay . To find that out, a human operator was asked to operate the system while packet loss and delay were varied. Each user had to perform the tasks illustrated in Fig. 2.

The challenge for each user was to guide a laser light into some targeted points by moving the robotic arm which had an attached laser diode. To do that, each user was asked to manipulate a joystick located at the other end of an unreliable wired-network. The test field was mounted on a white wall with small dark circles, known as targets. The targets were arranged in two different ways. In the first scenario the circles were equally spaced horizontally (see Fig. 2a), while in the second each circle was placed as shown in Fig. 2b.

Depending on how the joystick was operated the laser could move first along a horizontal direction and then in both horizontal and vertical directions. The user was first asked to point the laser light in each of the horizontally aligned targets. Then in a separate task, the light had to be pointed following the pattern from Fig. 2b . Whenever the laser light failed to hit a target, the user had to keep trying until the light overlapped the intended dark spot. For each experiment the mean TTCT was recorded under various conditions. QoE was quantified by the mean TTCT and by the mean opinion score (MOS) based on the user's qualitative rating of the system.

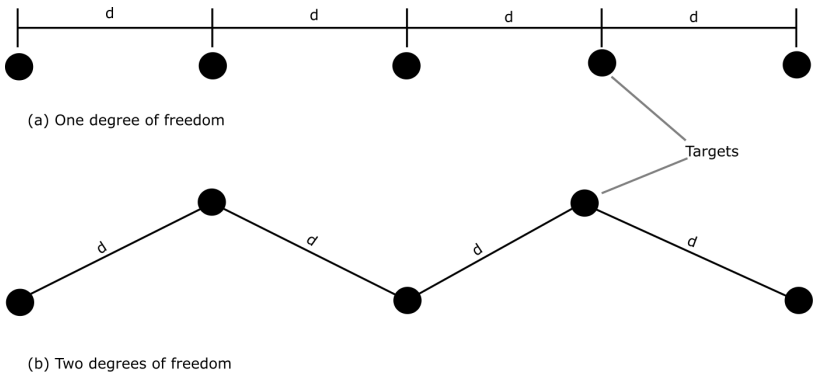

Figure 2: Test scenarios for evaluating the response of a robotic arm

To measure the change in the users' perception of quality three factors such as latency (ρ), packet loss (ν) and degrees of freedom (τ) were considered. This strategy enabled the construction of a model used to predict QoE. To characterize the model, packet delay was varied from $0ms$ and incremented by $100ms$ until a maximum value of $400ms$ was reached. The percentage of packet loss was initially set to 1% and incremented by 5% after each run of variation of packet delay as described above. The process was repeated until the packet losses reached 20%. The results showed the impact of each of the two metrics (packet variation of delay and loss) on both the task completion time and mean opinion score. A factorial design led to the prediction of a mean opinion score (MOS) expressed as

$$M = q_0 + q_\nu \iota + q_\rho \jmath + q_\tau \hbar + q_{\nu\rho} \iota \jmath + q_{\nu\tau} \iota \hbar + q_{\rho\tau} \hbar \jmath + q_{\nu\rho\tau} \iota \hbar \jmath \tag{1}$$

where q_0 stands for the average MOS. Also, the opinion score (OS) was defined as q_i with i being either a single factor or a combination of factors (ρ, ν and τ).
Additionally, a model based on multiple linear regression was proposed for M as:

$$M = b_0 + b_1 \nu_i + b_2 \rho_i \tag{2}$$

where b_0, b_1 and $b2$ are constant.

The result of the analysis based on factorial design showed that the changes in MOS were mostly affected by variations of the percentage of packet loss. In fact, the effect of packet loss was twice as important as the one caused by the variation of packet delay. The variation of packet delay causes a degradation of the QoE when its value is greater than $300ms$. Likewise a percentage of packet losses exceeding 10% degrades the QoE.

2.1.2 Quality of Experience from User and Network Perspective

In another related research, the correlation between network performance and end-users perception of quality was analyzed [6]. It is important to point out that quality was measured in terms of users' opinion score. The study also analyzed whether the frequency of utilization of a service by end-users could be viewed as a QoE factor. To find these answers two strategies were used. The first referred to as "classical comprehensive method" was an experimental approach in which measurements were performed on a constructed testbed. In contrast, the second approach called "automatic passive method" was a technique based on data collected from a real time user-network. Finally, results from the two approaches were compared before a conclusion was drawn.

2.1.2.1 Classical Comprehensive Method

This approach used the testbed shown in Fig. 3, whose elements are described below.

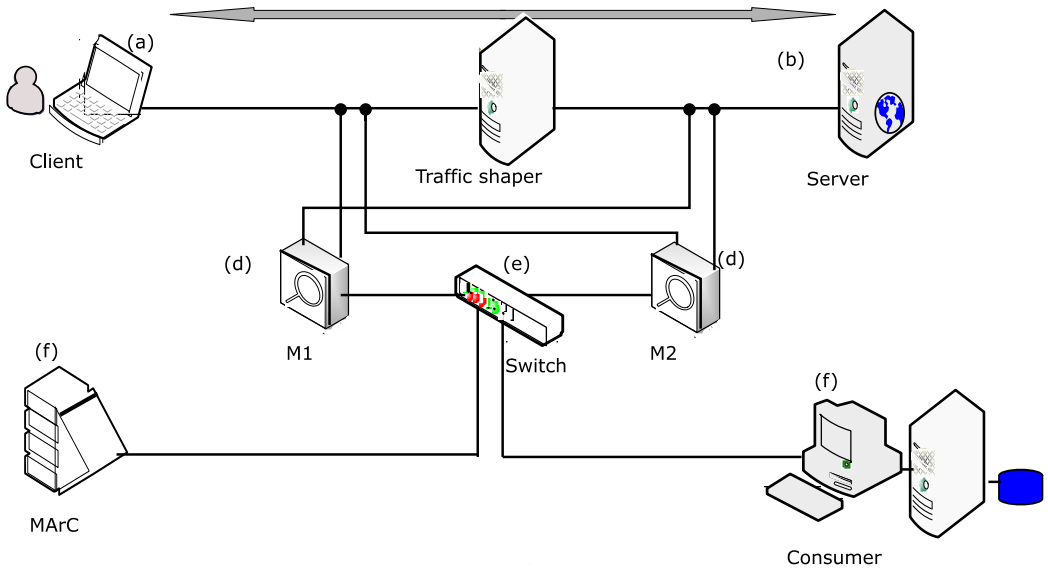

Figure 3: Test best for the experimental approach

- Workstation (a) symbolizes a client station used to represent the application level (i.e. user side). It is connected to the measurement points and to the traffic shaper where incoming and outgoing data transited.

- Device (b) is a traffic shaper (also called a Linux traffic controller) used to introduce packet losses at quantifiable rates. It is connected to the client, the measurement points and to the server.

- System (c) depicts a server station used to represent the network level (i.e. server side). It supplies and received data from the client.

- Units (d) symbolizes the measurement points (M2 and M3) used to record and process the traffic volume (i.e. data throughput) and timing (i.e. data download time) from either direction between the client and the server.

- Device (e) is an Ethernet switch used to interconnect the measurement points to the measurement area controller (MArC) and to the desktop.

- Stations (f) symbolize a MArC or a consumer desktop, both used to store and process data captured from both directions followed by the traffic.

During the experiment, packet losses were introduced in the traffic through the traffic shaper at a rate L. L was increased by two percent until a maximum value of ten percent was reached. For each level of packet loss, the user has to download a page of $X = 1.13MB$ ten times. Next a form, using the rating scale recommended by the International Telecommunication Union (ITU), had to be turned in by users to report their opinion score. The perceived QoE by users was analyzed based on regression methods. A correlation test was then conducted to determine whether the quality of service (QoS) parameters such as rate of loss L, download time T, and network data throughput R' were inter-dependent and if they were also related to users' perception of QoE.

It was noticed that T increased as L became larger. This led to a decrease in the application (i.e. on users' side) data throughput R. So T and R being both dependent on L were regarded as performance indicators from a user's perspective. To support this argument an exponential regression was used to predict

$$R' = 8.9exp(–0.25L) \tag{3}$$

and

$$T = 1.1exp(0.26L) \tag{4}$$

Moreover, a linear degradation of the users' OS was noticed as L exceeded a value of four percent. To investigate the relationship between QoE and QoS parameters, regressions on QoE with respect to R and R' showed that the users' interest in browsing dwindled when R' fell below a value of one megabit per second (Mps). Therefore QoE could be predicted as a function of both R' and R. To show this, a logarithmic regression predicted,

$$QoE = 1.5ln(R') + 1.153 \tag{5}$$

and

$$QoE = 1.2ln(R) + 1.3 \tag{6}$$

2.1.2.2 Automatic Passive Approach

The second approach known as an "automatic passive approach" relied on an asymmetric digital subscriber line (ADSL) core network architecture. The platform is illustrated in Fig. 4 with the description of its elements as follows:

- Computer (a) represents the customer station where data was downloaded and uploaded to the network.
- Set (b) is an access network used to serve thousands of customers.
- Device (c) represents broadband access server (BAS) used to capture the traffic from several digital subscriber line access multiplexers (DSLAM).
- Item (d) denotes a probe connected to the line between the BAS and the backhaul. It enables TCP/IP (Transmission control protocol/Internet protocol) headers to be captured in order to quantify a number of metrics characterizing the traffic (i.e traffic volume and number of packets) and the performance (data throughput and ratio loss).
- Set (e) is used to represent the backhaul of the network.
- Devices (f) depict a set of routers used to connect the backhaul to the Internet.
- Set (g) represents the cloud of Internet services.

To collect the data in sessions (i.e. time-frames) a threshold of 64 seconds was set to specify the acceptable duration for the traffic channel to remain in idle mode before the TCP connection dropped (i.e. session termination). A correlation was established between the computed traffic factor (i.e. session volume) and the performance characteristics. It was then discovered that loss ratio L affected the data throughput on the user's side as it increased or decreased. So the measurements were performed under two cases known as "session volume download" case and "session volume upload" case [6].

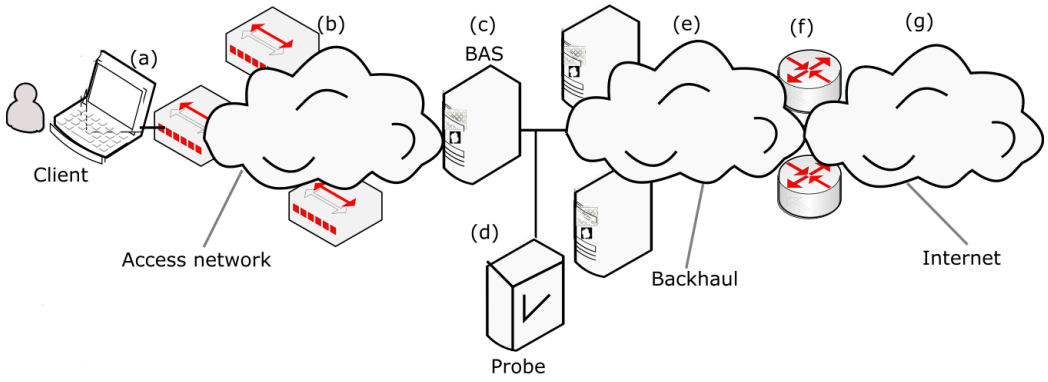

Figure 4: Collection infrastructure

During a session volume download, the size of the session was reduced as L exceeded $\frac{1}{1000}$. A power regression predicted the expression in Eq. 7.

$$V = 98L^{-0.62} \tag{7}$$

where V represented the session volume.

However, in the case of session volume upload, the value of L had to grow to approximately $\frac{4}{1000}$ before a decrease in V resulted. Again, to clarify this assertion a power regression demonstrated that V varied as shown in Eq. 8

$$V = 239L^{-0.42} \tag{8}$$

The results in the experimental approach were compared with those obtained in the passive strategy. Notice that in both approaches the network data throughput had to noticeably decrease before the users' perception of QoE degraded. Also when L decreased, both QoE and the data throughput increased [6].

2.1.3 ESA/NASA Projects

Another relevant research is the ongoing METERON project project announced in 2011. It is regarded as a complement of the human exploration telerobotics (HET), another research conducted by NASA.

Both aim at determining practical ways to operate robots from orbiting spacecrafts to perform missions in space locations inaccessible to humans. Experiments were performed in order to answer questions related to enabling exploration of many celestial bodies such as Mars, Moon etc., at a low cost.

METERON project intended to get started with the use of existing infrastructure to ensure strong communication and operation abilities, including innovation skills in robotics. These missions were controlled by the European Space Operation Center (ESOC)located in Germany [12] [2]. Figure 5 depicts the testbed used to conduct the experiments which roughly contains the following elements:

- A master located on the international space station (ISS). It is the main circuitry converting the operator's maneuver into digital instructions for the slave.
- A robust channel supported by delay tolerant network (DTN) nodes, real time control network (RTCN) nodes, antenna arrays and other network units in a way to provide a reliable communication.
- A slave(i.e. robot) located on the ground. It is a system that interprets commands received from the master.

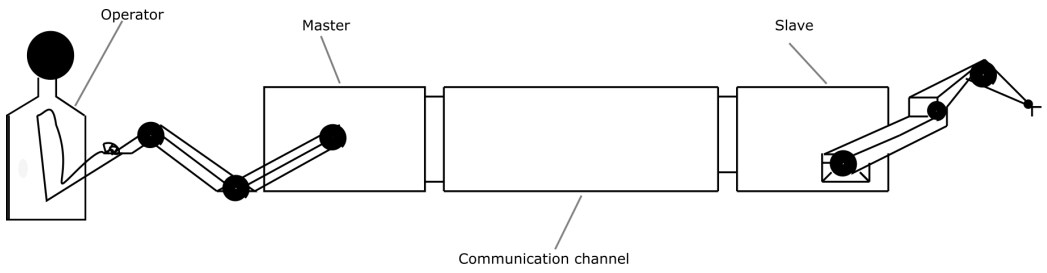

Figure 5: Test bed for the METERON experiment

To set up the communication channel terrestrial infrastructure was used. Therefore, an ISS simulator (i.e. an astronaut training field with simulated space variables) and a robot simulation facility (i.e. a robot training field similar to those found on other orbiting planets) had to be prepared. The aim was to test the effectiveness of communication protocols to support smooth interaction between parties not necessarily located on the same continent. As a result, the first experiment was entirely conducted with both the operator and the slave located on ground. Next, the operator located on the ground was replaced by an astronaut located on board the ISS, which is the only spacecraft that enables experiments with possible physical constraints such as microgravity [12].

Sending commands from a spacecraft at high altitudes to ground results in signal delay, fading and/or loss. The same situation happens when the spacecraft sets below the horizon where it is no longer in line-of-sight with the receivers on ground. Therefore in order to maintain communication under these circumstances a DTN, which can support space communications (i.e. interconnection of satellite nodes as part of DTN) is used to sustain the communication channel [2]. In fact ESA and NASA thought of DTN as a useful means to temporarily store packets on specific nodes when the network metrics do not allow a reliable data transfer from a master to a slave and viceversa. Consequently, commands sent by an operator in space to a robot on ground will only experience a delay when prompt data delivery is not possible. The same scenario applies when the receiver sends a feedback to the transmitter in space.

On the space station data traffic departs from and/or reaches the master platform through an ISS data bus. The bus supports DTN and RTCN protocols. One end of the ISS a data bus communicates with ground stations in Maspalomas in Spain, Villafranca in Italy, and Wellheim in Germany through line-of-sight parabolic antennas, while the other end links to a separate parabolic dish located on the spacecraft in order to send and/or retrieve information from another ground station at NASA White Sands in the USA through a satellite relay. Such a topology, also referred to as METERON operation environment (MOE), enables a human astronaut located on board the ISS to send commands to a robot located on ground using various DTN paths. Likewise, the robot has multiple route options to send a feedback to the sender [12] [2].

With the MOE set up, two communication oriented experiments referred to as OPSCOM-1 and OPSCOM-2 were conducted. In OPSCOM-1, the communication link between an on board ISS computer and a robot known as METERON operations and communication prototype (Mocup) was tested in October 2012. In this scenario, a research center in the USA had to authorize an on board ISS computer to send command to Mocup located in Germany (see Fig. 6). The goal was to test the efficiency and ability of DTN to support future implementations of METERON. The tasks comprised instructing Mocup to move forward and also send ground image snapshots to the USA as a feedback.

However given some limitations discovered in communicating from the robot back to the ISS, OPSCOM-2 had to be carried out in late 2014 to enable additional communication features such as simultaneous two-way data transfer via DTN, more data transfer capacity and capability to deal with missing data and managing data delivery. In the experiment, communication, remote operation and monitoring were all involved. Another robot known as Eurobot was tested on a $64m^2$ ground field. The field comprised small crater, boulder field, sandy dune and gravel slope area to make it resemble a real space environment for evaluating the performance of the robot under challenging conditions. The test lasted 90minutes. During the experiment an astronaut from an altitude of 400 kilometers on board the ISS had to drive Eurobot located in Netherlands. In addition, tests were carried by NASA where an astronaut located at an altitude of 300km in space had to send command to Eurobot on the ground [2]. In both cases, the robot in turn had to send data including video images in response back to the ISS for monitoring purpose.

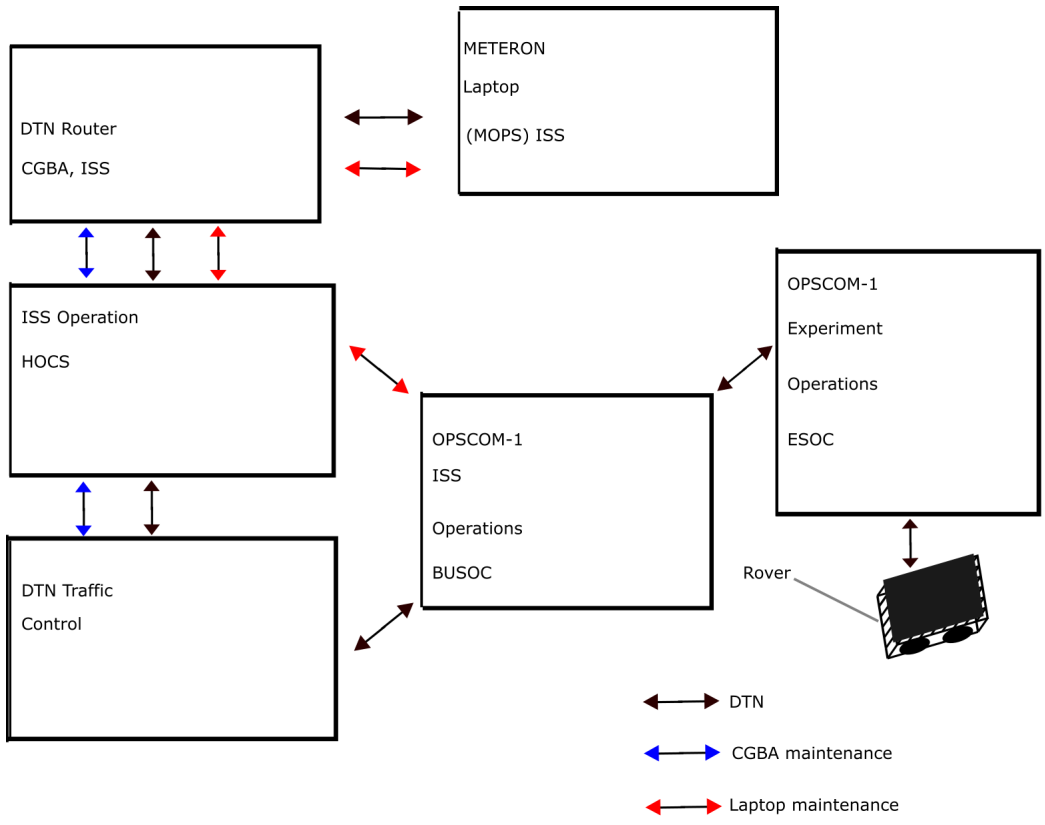

Figure 6: Testing the OPSCOM-1

Another experiment known as HAPTICS-1 was conducted by the end of 2014, where a robot arm (a.k.a. Skeleton-x-arm-2) was deployed on-board the ISS with the aim of enabling telepresence during robotic exploration on other planets [14]. Here the robot on field was equipped with sensors that allowed him to promptly interact with Skeleton-x-arm-2. Such a practice will enable astronaut user wearing Skeleton-x-arm-2 on-board the ISS to feel in real-time the conditions experienced by the robot on field.

In summary, previous studies, though conducted using different platforms, shared interesting points with regards to how QoE could be modeled as a quantifiable variable under unfavorable network conditions. However, none had proposed a solution to improve QoE. On the other hand ESA and NASA achieved the expected results out of the experiments performed in METERON/HET.

However, the use of DTN makes the system totally dependent on the network as a main tool for overcoming the effect of packet loss in order to maintain QoE.

2.2 Research Differentiation

This work is different from the previous research in the area for the following reasons:

1. Previous studies focused on quantifying metrics but did not include any compensation mechanism [5] [6].

2. Studies by ESA/NASA relied on DTN nodes contribution to maintain QoE. Such a strategy could possibly be challenged when run out of storage capacity or when packet storage time is out. On the other hand for future deployment of an autonomous robot known as Rollin'Justin, ESA implicitly intended to use an artificial intelligence approach to setup the experimental platform [2] [12]

In the work presented here, the network is not expected to play a role in the compensation process. Thus, no acknowledgement or storage units are used to detect packet losses or enable re-transmissions of lost packets. Additionally, the CPD still relies on incoming packets but strategically compensates for missing ones.

2.3 Background on Methods Proposed to Test the Hypothesis

2.3.1 Environment and State

2.3.1.1 Robot Environment

A robot environment is a medium that a robot interacts with in order to perform specific tasks. To do that, a robot needs to use at least one sensor used to report the state of the environment. However, the data reported is often affected with noise that might result from either extraneous environmental factors or robot activities (e.g. motors) or both. As a result this might lead to erroneous interpretation of data on the robot side. Therefore, in order to deal with such issues, a robot must base its analysis on inference (i.e. a preliminary assumption) [13].

2.3.1.2 Robot State

A state is a set of characteristics that are common to the robot and its environment. A state may be static (i.e. steady as time varies) or dynamic (i.e. changing with respect to time).

It may comprise information regarding the robot (e.g. motor speed, duty cycle, hardware status, etc.) and the environment itself (e.g. buildings surrounding the robot, weather, users etc.). Other independent variables such as Cartesian and angular coordinates, degrees of freedom of a robot arm etc., can be used to characterize a state.

A state enables a transition to another state when it is said to be *complete*. This means that all the required data is available to the previous state before the initiation of the transition to the next state. So no additional information is needed to make that transition. An important point to clarify here is that the next state does not necessarily have to be predicted by a deterministic dependent variables to confirm the completeness of the current state. In other words, the next state can also be a result of a stochastic process but will still be dependent on the current state. Such a process is referred to as a *Markov chain*.

Though it is also crucial to note that the requirement that a state at time $t = (T - 1)$ ought to be complete in order to predict the next state at time $t = T$ in practice this is often challenging. This is due to the fact that not all factors driving the state variables at both environment and robot levels are easy to quantify before the need for transition between states arises. Then, the state at $(T - 1)$ is said to be *incomplete*.

A state may be defined by both discrete and continuous variables. An example of continuous variable is the distance traveled between two cities. On the other hand, a joystick monitor checking whether the stick is released or not could be seen as a discrete variable. A state predicted on the basis of both discrete and continuous variables is referred to as *hybrid state*. Likewise, the transition of a state to another can be continuous or discrete (i.e. happening at discrete time frames).

2.3.2 Robot Communication with the Environment

Communication between the robot and the environment takes two basic forms. One strategy for a robot to interact with its environment is through observation. For instance, a robot can use its sensors to measure the status of its environment by collecting information (i.e. *measurement data*) such as power level of a radio signal, noise level, detection of packets in a wireless channel, etc. Such a process is called *measurement*.

Another way of communication is when the robot influences the status of the environment. For instance, a robot can use its hardware (e.g. motors) to change the state of its surrounding medium through task performance (so called *control actions*). However it is important to keep in mind that before a control action results, a measurement has to first take place. This means that the control action does not happen in practice in real-time given the delay between the collection of the measurement data and the occurrence of the control action.

A robot also needs to have the necessary information to know when transition from a state to another should take place. Such a data is referred to as *control data*. For instance, when moving, a robot might need to figure out the ideal time to go from a lower speed to a medium or a higher speed, and also determine the time-frame over which a speed should be maintained. Having said that, variables "speed" and "time" can be viewed as control data [13].

2.3.2.1 Stochastic Dependence

Over a time-frame $[T_x; T_y]$ where $T_x \leq T_y$ a robot quantifies both the measurement data $M_{T_x:T_{y-1}}$ and control data $k_{T_x:T_y}$ in preparation for the transition from a state to another state. The measurement data collected from T_x to T_{y-1} and the control data data from T_x to T_{y-1} are be respectively expressed as based on the assumption that $[T_x; T_y]$ could be broken down into smaller time-frames. However, for the sake of simplicity it is assumed that only one control data will be considered at a time step $T \epsilon [T_x; T_y]$. For example, the time interval $[(T-1), T]$ will be used to describe the change in only one control data k_T ϵ $k_{T_x:T_y}$. Again the transition from a state s_{T-1} at time $(T-1)$ to a state s_T at time T is stochastic. Therefore predicting s_T from $(T-1)$ is a result of a probability distribution function p as in

$$M_{T_x:T_{y-1}} = M_{T_x}, M_{T_{x+1}}, M_{T_{x+2}}, M_{T_{x+3}}, ..., M_{T_{y-1}} \tag{9}$$

and

$$k_{T_x:T_y} = k_x, k_{T_{x+1}}, k_{T_{x+2}}, k_{T_{x+3}}, ..., k_y \tag{10}$$

$$p(s_T \mid s_{T_x:T-1}, M_{T_x:T-1}, k_{x:T}) \tag{11}$$

If the state s_T is complete, the measurement data, the control data and the state at a time step $(T-1)$ will then be considered to be a summary of $M_{T_x:T-1}$, $k_{T_x:T}$ and $s_{T_x:T-1}$, respectively. Therefore, the state s_{T-1} and the control data k_T can sufficiently predict state s_T at time T [13]. Hence,

$$p(s_T \mid s_{T_x:T-1}, M_{T_x:T-1}, k_{T_x:T}) = p(s_T \mid s_{T-1}, k_T) \tag{12}$$

Chapter 3: System and Experimental Design

3.1 Research Description

3.1.1 Goal

This work aims at compensating for packet losses that may impact the QoE perceived by a human operator when remotely operating a CPD. In this study it is hypothesized that *"it should be possible, under unfavorable network conditions, to compensate for QoE by offsetting the effect of packet losses in the path between the controller to the CPD."*

3.1.2 Research Question

How can a packet loss compensation algorithm be designed in such a way that it aids in maintaining the QoE?

3.1.3 Requirements

In order to answer the research question the following elements are required.

- An experimental testbed which includes a remotely controlled CPD.
- A selection of tasks to be performed by a human operator while controlling the CPD.
- A method to artificially degrade the wireless link. This method is needed since, for practical purposes, the CPD in the same room as the operator and thus the link introduces little or no losses.
- A compensation mechanism. This mechanism should help maintain the QoE when packet losses are detected.

3.1.3.1 Experimental Testbed

Figure 7 shows the testbed used for this project with its five main elements as described next.

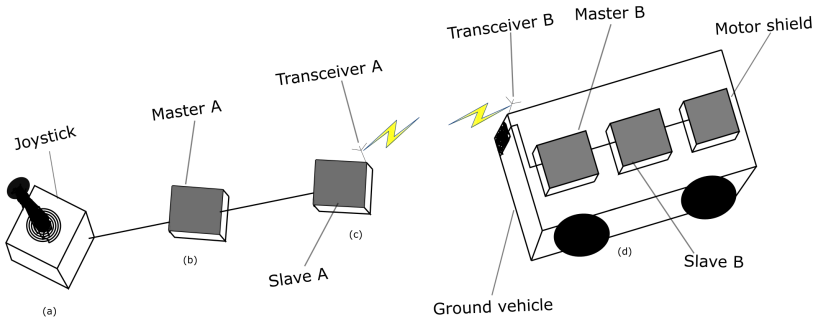

Figure 7: Experimental testbed for testing QoE compensation mechanism

- Item (a) is a joystick operated by the user to move the CPD.
- Item (b) is a first electronic board (i.e. master A) where the analog input from the joystick is converted into a digital signal (i.e. commands).
- Item (c) depicts a second electronic board (i.e. slave A) connected to master A and to a wireless transceiver (transceiver A). The link emulator used to simulate packet losses is also embedded here.
- Unit (d) represents the CPD, which is set to receive instructions from the controller (i.e. the networked joystick, master A , slave A and transceiver A) through another wireless transceiver (i.e. transceiver B) attached to it. Two electronic boards, master B and slave B are mounted on top of the vehicle. Master B controls the radio receiver and connects to slave B. In turn, slave B instructs the motor controller to drive the four (4) motors located inside the vehicle.

3.1.3.2 Experimental Tasks

Three different tasks were designed to test the system as shown in Fig. 8. These tasks share similar design principles with those proposed in the past to test human-computer interfaces [9]. Participants were recruited to test the system. Each participant was required to perform the three tasks as described next.

a. The first task consisted of a rectilinear path (see Fig. 8a) where each participant had to move the robot from the beginning of the path to the end without stopping the vehicle. This is later referred to as “task 1”.

b. The second task consisted a rectilinear trajectory of the same dimensions as the one in Fig. 8a, but with stopping points spaced one meter apart (see Fig. 8b). Each participant was required to move and stop the car exactly at each of the perpendicular lines along the path. Whenever a participant missed a stop he or she was required to keep trying until getting it right before proceeding to the next line. Such a task is later referred to as "task 2".

c. In the third task, later referred to as "task 3", subjects were required to move the CPD on a sinusoidal path over a distance of 6 meters (Fig. 8c). Participants had to drive the CPD from the beginning of the path to the end while trying to avoid unnecessary stops. For each experiment, the time to complete tasks (TTCT) was recorded along with the link condition, the task number and the compensation status (i.e. with or without the compensation mechanism). The link condition indicates the quality of the connection and will be discussed in the next subsection.

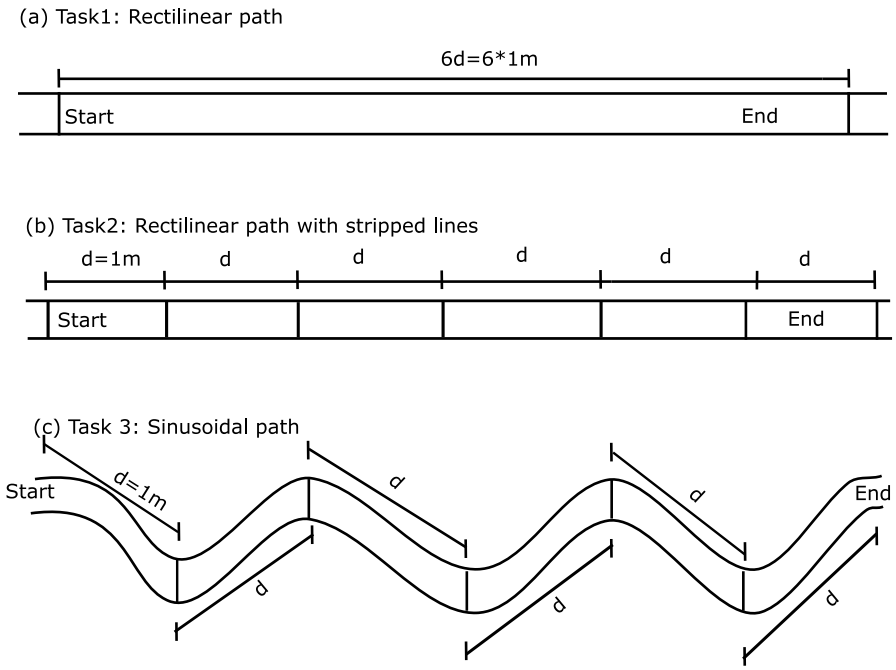

Figure 8: Task trajectories for the CPD

3.1.3.3 Wireless Link Degradation

The experiment relied on experiencing packet losses between the controller and the CPD. However, because the controller and the CPD were placed in the same room there were none or very few packet losses over the wireless channel. Therefore, it was first necessary to introduce artificial packet losses. To do this it was first crucial to understand and characterize the packet loss process of the wireless channel used in this study.

In order to determine how to simulate packet losses over the radio interface, the radio channel was characterized with a two state Markov model [11]. In the model the states are usually referred to as up and down. In the up state it is assumed that no packet losses occur, while in the down state all packets are lost. This model has been shown to provide a good approximation for packet losses over different wideband and narrowband channels.

3.1.3.4 Channel Characterization

The purpose of the channel characterization is to determine the mean durations $1/\alpha$ and $1/\lambda$ of the up and down states of the channel models respectively as depicted in Fig. 9. The variables α and λ are rates. They define the duration over time that the link should remain in the up and the down state respectively. The values of α and λ are defined as shown in table 1.

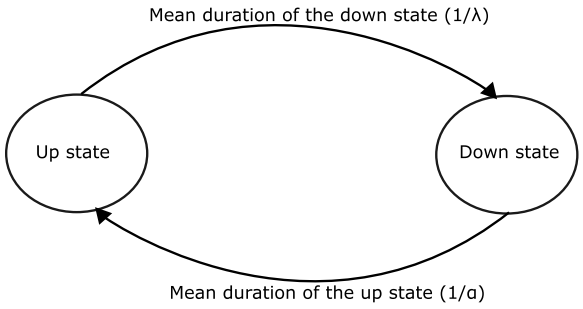

Figure 9: Two-state transition diagram

Table 1: Mean duration (in millisecond) of the up state (i.e. $1/\alpha$) and the down state (i.e. $1/\lambda$) used for each link condition

Link condition	$1/\alpha$ **(ms)**	$1/\lambda$ **(ms)**
Very good	99	1
Good	84	16
Meduim	50	50
Bad	38	62
Worst	11	89

Table 1 presents the values of the mean durations of the up state (i.e. $1/\alpha$) and the down state (i.e. $1/\lambda$). These values are deduced from Tables 2 and 3 ahead. For example, from the Tables, with the CPD located five metres (i.e. in very good link condition) from the transmitter the mean durations of the up state and the down state are 181762.2ms and 7.2ms respectively. Therefore, these values can also be translated into a mean duration of the up state of 99ms and a mean duration of the down state of 1ms for every duration of 100ms (see Table. 1). The same rule applies when the CPD is located 10, 15, 20, 25m away from the transmitter.

To experimentally determine the mean duration of each state of the channel a transmitter and a receiver were placed at varying distances from each other. Then the transmitter was set to number each packet and send it to the CPD at the maximum rate allowed by the hardware (i.e. every 3.6 milliseconds). For example, when a packet is sent to the CPD with an attached number $N = n$, the next packet has to be sent with the number $N = n + 1$ and so on. Such a procedure helped figuring out the distribution of the number of consecutive missing packets. Depending on the distance and obstacles between the transmitter and the receiver, packets might get lost in long or short runs of packets. Each experiment employed approximately $100,000$ packets and multiple runs.

Next, the received and missing packets were analyzed separately. The average sequence length of received packets was used to characterize the average duration of the up state of the channel, while missing packet sequences were used to characterize the down state.

When the CPD was located less than 5 meters of the controller, all the packets were successfully received since the link remains in the up state. At 5 meters, only four (4) missing packets out of $100,000$ were recorded.

The average duration of the up and down states was respectively estimated and plotted when the CPD was at 10, 15, 20 and $25m$ away from the transmitter. The distributions of the received and missing packet sequences are shown in Fig. 10 through 15.

Notice that the inset in every figure shows distributions with long tails. Therefore it was necessary to identify outliers before computing the mean state durations. A data (D) value was considered an outlier when either of the following conditions is met: $D < [Q1-(1.5*QI)]$ or $D > [Q3 + (1.5 * QI)]$. Where QI is an interquartile range, $Q1$ is the first quartile and $Q3$ is the third quartile.

Tables 2 and 3 show the statistics for the up and down states distributions. Notice how an increase in the separation distance between the transmitter and the receiver results in variations of the state duration. In both tables the values of the standard deviation reflect a positively skewed distribution of the duration of run of packets for both the up and the down states distributions.

Notice that the mean values in table 2 and 3 describe the values of $1/\alpha$ and $1/\lambda$, respectively. The highest value of $1/\alpha$ and the lowest value of $1/\lambda$ were achieved when the CPD was ten meters away from the transmitter given that only two packets were missing. On the other hand, the highest value of $1/\lambda$ and the lowest value of α were achieved when the CPD was 25m away from the transmitter.

Table 2: Up state duration (in milliseconds) statistics for different distances between the transmitter and the receiver

Parameters	**5m**	**10m**	**15m**	**20m**	**25m**
Mean	181762.2	158.4	35.3	42.1	12.1
Median	181762.2	39.6	18	21.6	12.3
Mode	NA	7.2	10.8	7.2	7.2
Std Deviation	40009	268.5	103.8	64.9	79.31
Range	56581.2	1936.8	1112.4	1072.8	1311.1
Minimum	153471.6	7.2	7.2	7.2	7.2
Maximum	210052.8	1944	1119.6	1080	1320
Sum	363524.4	207381.6	35510.4	92437.2	112970
Count	2	1309	750	2193	1924

Table 3: Down state duration (in milliseconds) statistics for different distances between the transmitter and the receiver

Parameters	**5m**	**10m**	**15m**	**20m**	**25m**
Mean	7.2	30.2	33.3	66.5	98.4
Median	7.2	7.2	7.2	7.2	7.2
Mode	NA	3.6	3.6	3.6	3.6
Std Deviation	5.1	88.4	297.6	186.9	311.42
Range	7.2	2008.8	8512	4305.6	13230.2
Minimum	3.6	3.6	3.6	3.6	3.6
Maximum	10.8	2012.4	8515.5	4309.2	13235
Sum	14.4	63957.6	50571.5	256734	327021.2
Count	2	2116	1516	3861	3162

From Tables 2 and 3 the values of $1/\alpha$ and $1/\lambda$ represent respectively the means of the up state and the means of the down state for each range between the transmitter and the CPD. In addition the value of $1/\alpha$ and $1/\lambda$ in Table 1 are derived from the Means in Tables 2 and 3. The variable " Count" in Tables 2 and 3 represents the number of transitions to the up state and to the down state, respectively.

Figures 10 through 15 show histograms of the length of the sequence of received and missing packets at different separations in between the transceiver and the receiver.

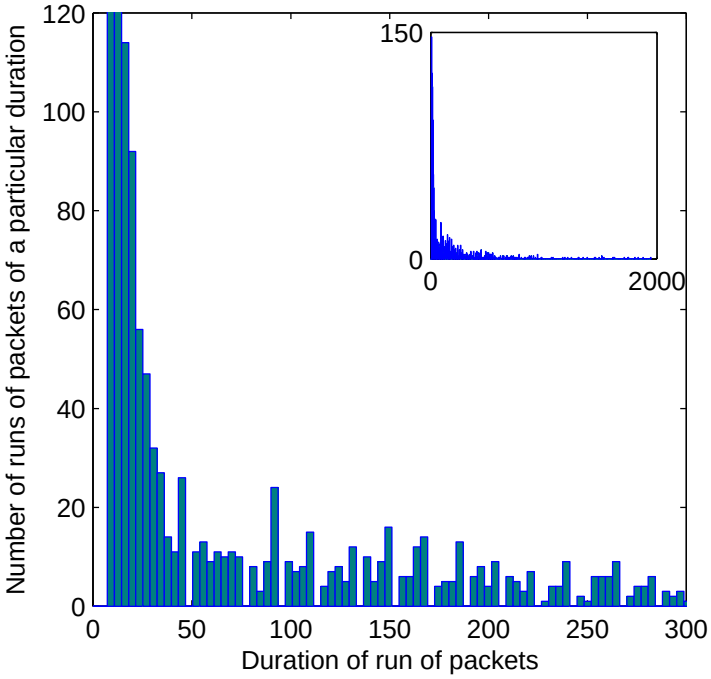

Figure 10: Up state at 15 meters

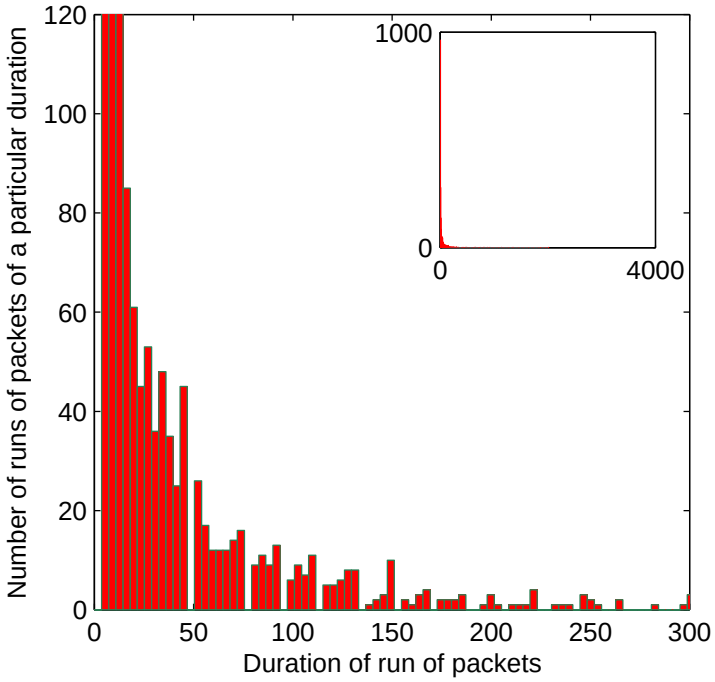

Figure 11: Down state at 15 meters

Fig. 10 and Fig. 11 show similar trends. Both figures show a rapid descent of the number of runs of packets of a particular duration. It is much more common to have runs of packets of short duration than runs of packets of long duration.

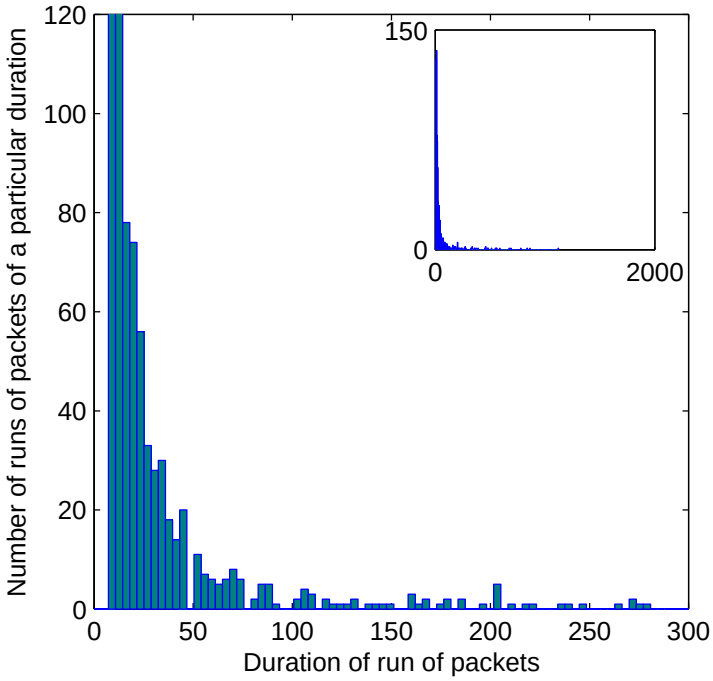

Figure 12: Up state at 20 meters

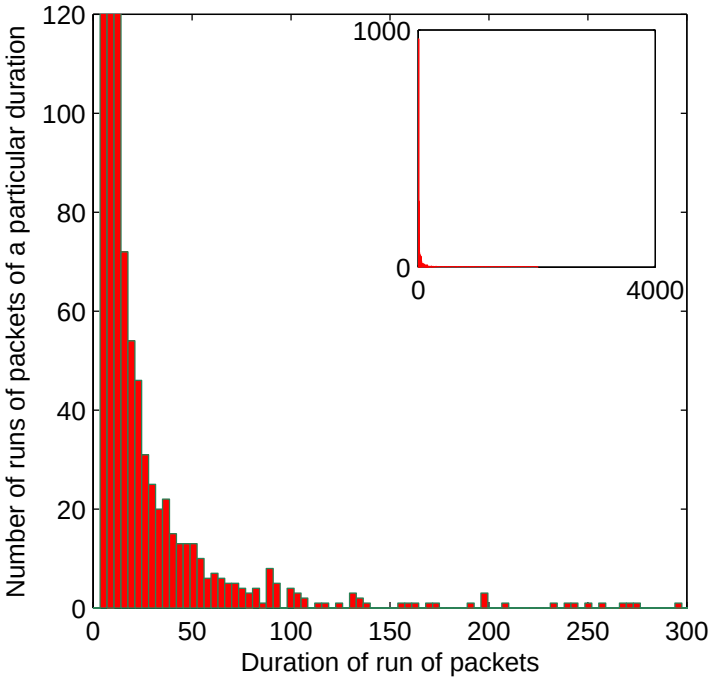

Figure 13: Down state at 20 meters

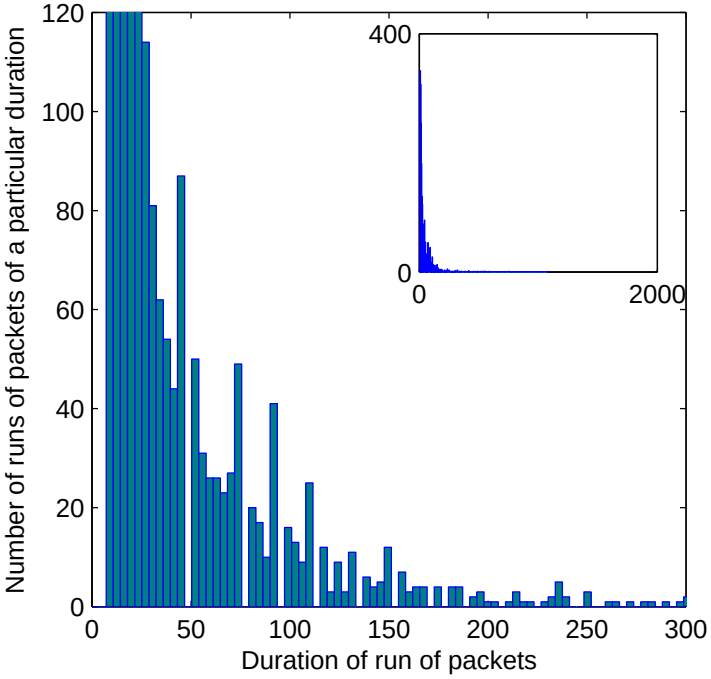

Figure 14: Up state at 25 meters

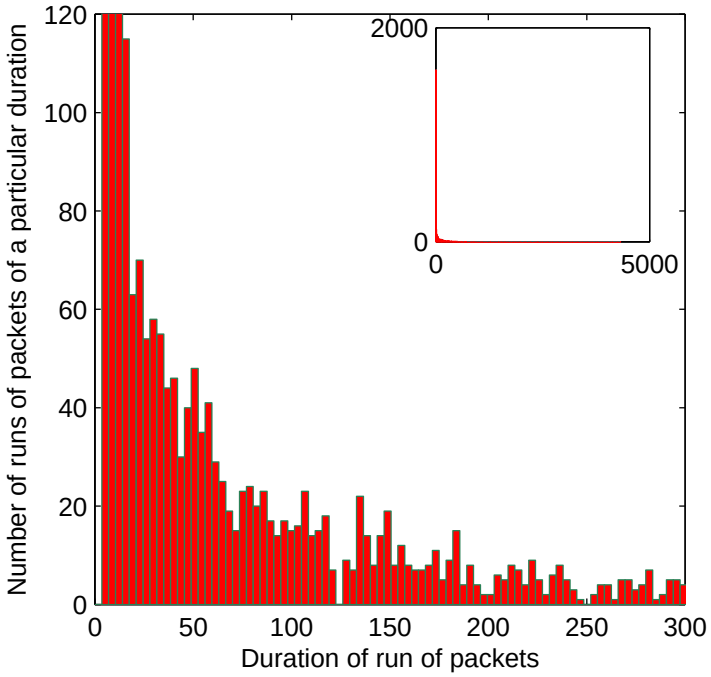

Figure 15: Down state at 25 meters

The channel characterization method provided the mean durations of the up and down states for the wireless link. Additionally for each separation between the transmitter and the receiver the percentage of packets lost was computed and labeled as shown below.

- "Perfect", with the percentage of received packets equal to 100. This was achieved when the CPD was located at less than 5 meters from the controller.
- "Very good", with the percentage of received packets greater than or equal to 97 but less than 100. The very good link condition was achieved when the CPD was located at 5 meters from the controller.
- "Good", with the percentage of received packets greater of equal to 75 but less than 97. A good link performance was recorded when the CPD was at 10 meters from the controller.

- "Medium", with the percentage of received packets greater of equal to 50 but less than 75. The medium link performance was achieved when the CPD was located at 15 meters of the controller.

- "Bad", with the percentage of received packets greater or equal to 25 but less than 50. The link performance was said to be bad when the CPD was at 20 meters from the controller.

- "Worst", with the percentage of packets received packets below 25. As the CPD was at 25 meters from the controller, the link condition became worst.

3.1.3.5 Compensation Mechanism

The goal of the compensation mechanism is to maintain the QoE when packet losses occur over the wireless link between the controller and the CPD. In this research two types of packets are sent to the CPD. One comprises moving packets that are generated when the user operates the joystick (moving packets instruct the vehicle to move in a specific direction). In the discussion ahead, moving packets will be referred to as just "packets". The other type involves idle packets which are generated for a short time (i.e. over 50ms) when the user releases the joystick. The choice of the short period of transmission of idle packet is made based on the assumption that the unfavorable channel condition may cause the loss of several idle packets. Therefore it is assumed that at least one idle packet will be received by the system.

A variable known as *idle controller* (ς) is defined to quantify the status of the joystick. ς is set to 1 when an idle packet is received by the CPD and to 0 when the joystick is active. Note that idle packets neither instruct the CPD to move nor to stop. They only report the status of the joystick for controlling a variable (φ) which will be discussed later. In the discussion ahead, moving packets will be referred to as "packets".

The initiation of the compensation process is based on the information regarding packet loss. When packet loss occurs, the system is required to continue its previous tasks for a short period of time known as *compensation time* (ε_e). ε_e in turn depends on the number of packets received during the period that the joystick remains active (i.e. when $\varsigma = 0$).

The size of each packet is 32 bytes.

All packets are transmitted to the CPD at a fixed rate of 71Kbps. When a packet is sent to the CPD, the shortest period before transmission of another packet is 3.6ms (the maximum rate allowed by the hardware). The receiver detects packet losses by analyzing sequence numbers sent in the packets payload. The following steps are considered in designing the mechanism.

A variable referred to as *control instructor* (φ) is defined to track and maintain the state of the system when packet loss occurs. This variable is initially set to 0 before packets arrival at CPD. The value of φ increases by one for every packet received. φ determines also the compensation time ε_e which is the time the system continues the task it was performing prior to the occurrence of packet loss. Fig. 16 shows the mapping of $\varepsilon_e \in \Upsilon$={ 300, 700, 1000, 1500ms} as a function of φ. The figure shows that the shortest compensation time is used when $\varphi \leqslant 3$, while the longest compensation time is used when $\varphi > 10$. These values were determined heuristically after some initial trials.

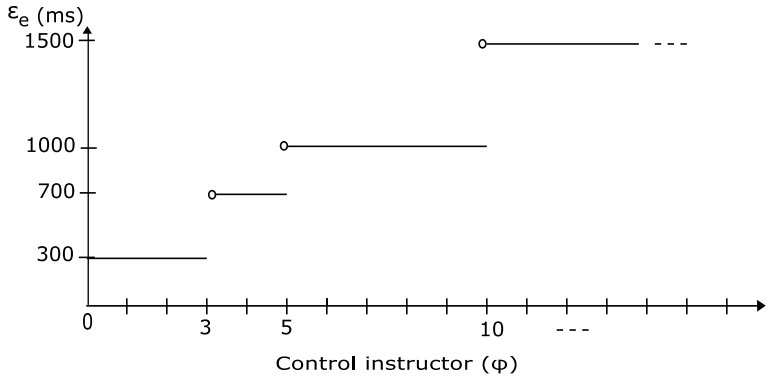

Figure 16: Mapping function, $\varepsilon_e(\varphi)$

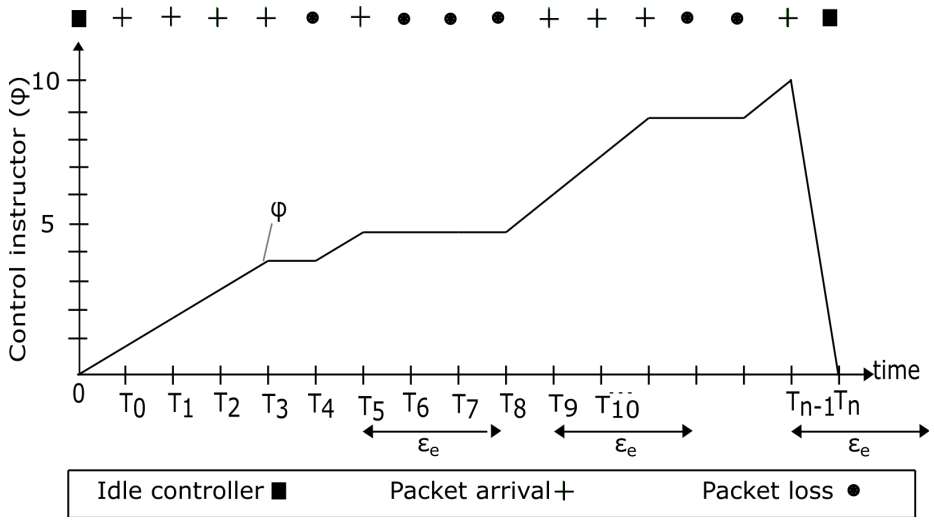

Figure 17: Example of the evolution of φ with time

The evolution of φ over time as well as its impact on the state transition diagram are illustrated in Fig. 17 and 18, respectively. In Fig. 17 three elements represent the input to the system. These comprise the idle controller (■ symbol means that the joystick goes idle), the moving packets arrival (+ symbol) and the moving packet loss (• symbol). The solid curve in the figure represents the variation of the control instructor with time t.

- At $t = 0$ the joystick is idle (■ symbol). Hence the value of ς is set to 1, which resets the value of φ to 0.
- From $t = T_0$ through $t = T_3$ packets are received respectively, increasing φ by four.
- At $t = T_4$ a packet loss occurs. The system detects no packet. Therefore, the value of φ remains constant (i.e. φ remains at four since no new packet is received).

- At $t = T_5$ a packet arrives, causing φ to increase by one. However the arrival of the new packet enables the system to compute the packet loss and find out that a packet loss occurred at T_4. Therefore, it initiates a compensation cycle corresponding to the value of φ. As shown in Fig. 16, the compensation time ε_e=700ms is required.

- From $t = T_6$ to $t = T_8$ three packets are missing, so the value of φ remains at five since no new packet is received.

- At $t = T_9$ a packet is received. Consequently, the system finds out that packet losses occurred from T_6 through T_8. Hence compensation is initiated over ε_e=1000ms (the value of φ is now equal to six). During each session of compensation the value of φ remains constant.

- At $t = T_{n-1}$ a new packet arrives, increasing φ to nine, and initiating compensation, by replicating the missing packets over a duration ε_e=1000ms.

- At $t = T_n$ the user has released the joystick, which sets the value of φ to 0.

The state transition diagram shown in Fig. 18 depicts how change in state of the system occurs as a function of φ. This concept which is similar to that of a finite state Markov model can be explained as follows.

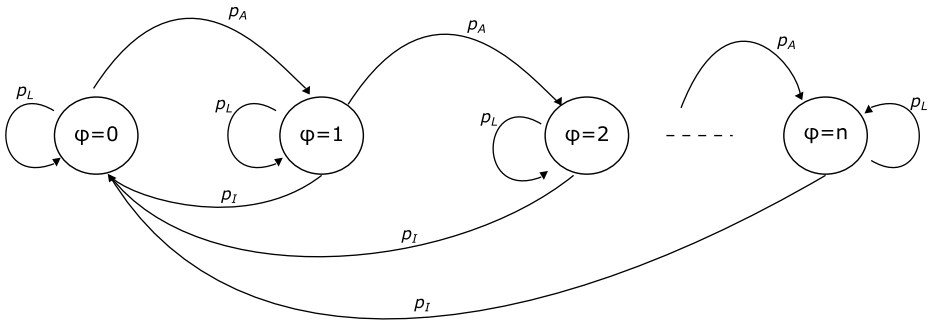

Figure 18: State transition diagram. φ represents the state of the system and varies when a packet is received

a. The value of φ is initially zero. It then increases by one when a packet is received. Though, it is also worth mentioning that the increase in φ is stochastic since it depends on the probability of packet arrival, p_A.

b. φ remains constant whenever a packet loss occurs. Packet losses are also probabilistic and are denoted by p_L. Transition of the system to the next state depends on the value of φ. Therefore, the system remain in the same state if the value of φ does not change.

a. At any time the user might release the joystick in order to efficiently control the CPD. Whenever the joystick goes idle, the value of φ become zero taking the system back to the original state regardless of its previous state. Like all moving packets, idle packet arrival to the system is probabilistic and is referred to as p_I.

This thesis intended to compensate only for packet losses. However experiments revealed that when the CPD moves away from the transmitter, packets experience delay. Likely this occurs due to internals of the radio design. Packet delay means that the period of interpacket arrival becomes greater than 3.6 ms. In order to tell whether a packet delay has occurred the duration of each inter-packet arrival is timed. For each packet received, the elapse time is measured.

To approach this problem, a new variable referred to as *traffic controller* (ϱ) is introduced. Figure 19 illustrates how ϱ operates. The value of ϱ is initially set to zero. This value increases by one for each packet received. However, ϱ returns to zero whenever delay occurs but remains constant when packet loss occurs or when the user releases the joystick. Notice that the occurrence of packet delay is also probabilistic and is referred to as p_D in Fig. 19.

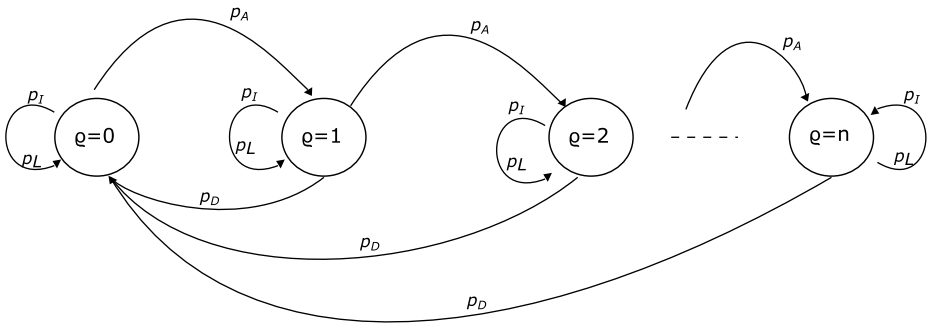

Figure 19: Mode of operation of ϱ

The approach used for compensating for packet delay requires segmenting the set of received packets into bursts (group of packets) in between delay occurrences. The count for each burst starts after a delay is experienced. Also, whenever delay occurs, ϱ become zero. This concept is depicted in Fig. 20 where delay ($\sim$ symbol) is introduced in addition to the input variables used in Fig. 17. The curve in dotted line illustrates the variation of ϱ with time. This helps track the size (number of packets) in each burst in-between packet delay occurrence. For instance, from Fig. 20 the first burst lasts from $t = T_0$ to $t = T_3$ and counts four packets before a delay occurrence at time $t = T_4$. However, delay detection by the system is only possible with a new packet arrival. For instance, the first delay is detected at T_5 even though it occurred at T_4.

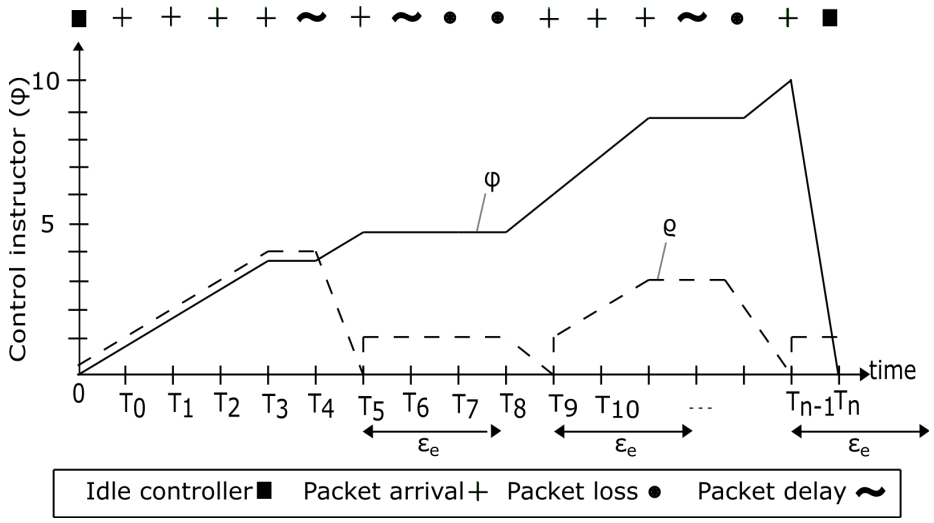

Figure 20: Using ϱ to determine the size of runs of packets in-between delay

In order make ϱ useful to the system the following steps are considered.

a. Since ϱ increases by one for each packet received, it is then possible to count the number of packets in each run of packet through the value of ϱ in-between packet delay.

b. Bursts that contain less than 300 packets are grouped in two types. Type 1 comprises runs of packets with less than 20 packets each, while type 2 comprises runs of packets with more than 20 packets.

c. Whenever delay occurs after a run of packets of type 1, the compensation system check the value of φ and compensation is performed by the system as discussed earlier depending on the value of φ.

d. For runs of packets of type 2, it is noticed that delay is less severe on the system performance. This means that less compensation time is required. Therefore when delay occurs following a run of packets of type 2, compensation takes place over a duration $\delta =$ 100ms.

e. It was noticed through the serial monitor that packet delay occurs less frequently when runs of packets in-between delay contain at least 300 packets each. Therefore, it was assumed that such a delay could be tolerated without significantly affecting the performance of the system.

As depicted in Fig. 21 the system uses ε_e to compensate for packet delay when ϱ is less than 20. When the value of ϱ is between 20 and 300, a fixed compensation time of 100ms is provided for each packet delay. When the value of ϱ goes above 300, any delay is ignored by the system. However, a delay causes the value of ϱ to immediately return to zero.The variables mentioned before along with their respective modes of operation are summarized in Table 5, where each variable is identified as a dependent variable (DV) or an independent variable (IV).

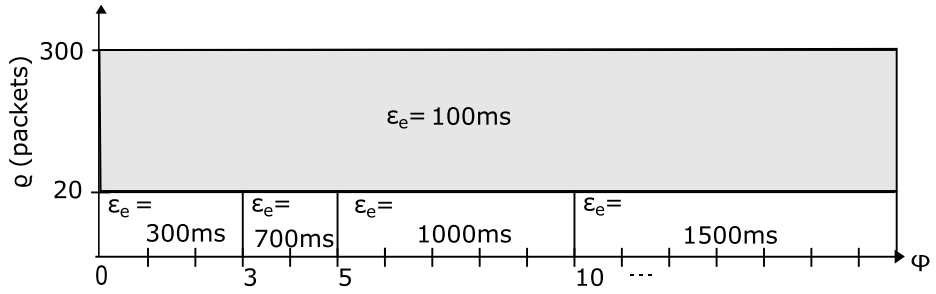

Figure 21: Mapping function, $\varepsilon_e(\varphi, \varrho)$

Table 4: Variables name and operation

Variable	Operation
ϱ	Goes to 0 when packet delay occurs
ϱ	Maintains its current value when packet loss occurs
ϱ	Maintains its current value when the joystick goes idle
φ	Goes to 0 when the joystick is idle
φ	Maintains its current value when packet loss occurs
φ	Maintains its current value when packet delay occurs
ς	Controls φ
ς	Sets to 1 when the joystick is idle φ
ς	Resets to 0 when the joystick is active φ
ε_e	Determines the duration of the compensation cycle
p_A	Defines the chance of receiving a packet
p_L	Determines the chance of losing a packet
p_I	Determines the chance of receiving an idle packet
TTCT	Quantifies QoE

Table 5: Variables name and type

Variable	Name	Type
ϱ	Traffic counter	IV
ϱ	Traffic counter	IV
ϱ	Traffic counter	IV
φ	Control instructor	IV
φ	Control instructor	IV
φ	Control instructor	IV
ς	Controller idle	IV
ς	Controller idle	IV
ς	Controller idle	IV
ε_e	Compensation time	DV
p_A	Probability of packet arrival	IV
p_L	Probability of packet loss	IV
p_I	Probability of Idle	IV
TTCT	Mean time to complete tasks	DV

Chapter 4: Discussion of Results

This section describes how experiments were conducted to evaluate the performance of the compensation mechanism. For example, a t-test interestingly shows how the percentage decrease in TTCT maintained an ascending trend as the percentage of received packet decreased. The limitations of the system are addressed next.

4.1 Statistical Analyses

As stated earlier, testing the efficiency of the compensation mechanism required the use of an approach similar to "one-group pretest-posttest design". In this thesis instead of a group of subjects, a single CPD is pretested before implementation of the compensation system and then post-tested after implementation of the system.

Approximately 60 participants were recruited through a convenient sampling in order to evaluate the system. The recruitment process started with a distribution of fliers to a targeted group of students and non students on campus irrespective of their age, gender, sex and religion. Each participant was given 15 minutes to practice before performing three tasks as stated earlier, for each link condition (i.e. perfect (prf), very good (vgd), good(gud), medium (med), bad (bad) and worse (wrs)).

The performance of the CPD was first tested without compensation, and then with compensation. In either case the mean TTCT was recorded. TTCT is represented by blue squares and red circles as depicted in Fig. 22 through 26. Every task was performed in all the link conditions and was repeated twice (i.e. with and without compensation). For each task, the red circles represent the mean TTCT for the uncompensated system, while the blue squares represent the mean TTCT for the compensated system. The resulting error rate associated with each task performance was represented in terms of errorbars attached to the respective squares and circles.

In each figure a green line is used as a reference. The green line represents the TTCT of the system when the link was assumed to be in a perfect (i.e. with 0% packet lost) condition.

It was noticed that in very good link conditions, it is hard to differentiate the blue squares form the red circles (see Fig. 22).

However as the link conditions respectively degraded from very good to good, medium, bad and worse, the blue squares became gradually distinct form the red circles (see Fig. 23 through Fig. 26).

In order to provide a formal explanation to this a t-Test was run in each case. Tables 6 through 10 describe a t-Test analysis carried out with regards to the $\mathrm{TTCT}_{\Phi\Psi\_\eth}$, where Φ, Ψ and $\eth$ respectively stand for task number, the link condition and the system status (i.e. compensated(c) or uncompensated(u)).

a. In very good conditions the TTCT before and after compensation did not significantly vary. In t-Test analysis shown in table 6 the value of "t Stat" is less than that of "t Critical two-tail". As a result, no major contribution from compensation system was noticed.

b. In contrast in Fig. 23 through 26 the blue squares are clearly separate from the red circles. As the link conditions degraded, the gap between blue squares and red circles also becomes larger. In order to confirm the statistical significance of these gaps, sessions of unpaired t-Tests were conducted in each link condition as shown in Tables 7 through 10. Recall that the aim is to examine the variations in TTCT before and after the implementation of the compensation mechanism. In all scenarios the value of "t Stat" is by far greater than that of "t Critical two-tail". Consequently, the compensation algorithm played a significant role in reducing the TTCT when the link experienced a percentage loss above three percent (3%).

Table 6: t-Test in the very good link conditions

	$\mathbf{TTCT}_{1_{vgd\_u}}$	$\mathbf{TTCT}_{1_{vgd\_c}}$	$\mathbf{TTCT}_{2_{vgd\_u}}$	$\mathbf{TTCT}_{2_{vgd\_c}}$	$\mathbf{TTCT}_{3_{vgd\_u}}$	$\mathbf{TTCT}_{3_{vgd\_c}}$
Mean	12.82	12.81	15.90	15.64	12.45	12.44
Var	0.49	0.73	1.80	1.48	1.06	1.13
t Stat	0.03		0.71		0.05	
$p(T <= t)$ one-tail	0.48		0.24		0.48	
t Critical one-tail	1.67		1.68		1.68	
$p(T <= t)$ two-tail	0.97		0.48		0.96	
t Critical two-tail	2.01		2.01		2.01	

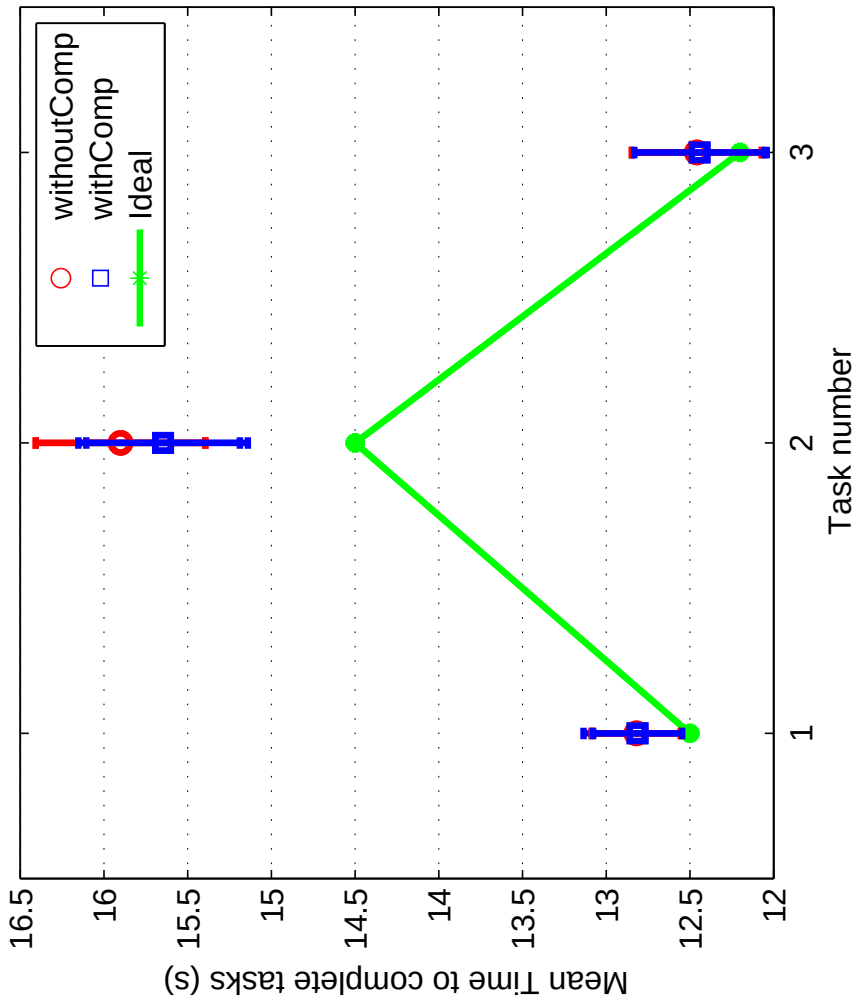

Figure 22: Mean TTCT in the very good link condition with a confidence level of 95%.

Table 7: t-Test in the good link conditions

	$\mathbf{TTCT}_{1_{gud\_u}}$	$\mathbf{TTCT}_{1_{gud\_c}}$	$\mathbf{TTCT}_{2_{gud\_u}}$	$\mathbf{TTCT}_{2_{gud\_c}}$	$\mathbf{TTCT}_{3_{gud\_u}}$	$\mathbf{TTCT}_{3_{gud\_c}}$
Mean	17.46	14.57	21.04	17.83	17.62	14.98
Var	2.07	0.98	2.66	2.99	2.47	1.99
t Stat	8.44		6.89		6.37	
$p(T <= t)$ one-tail	4.7E-11		4.5E-09		3.1E-08	
t Critical one-tail	1.68		1.68		1.67	
$p(T <= t)$ two-tail	9.5E-11		9.1E-09		6.2E-08	
t Critical two-tail	2.01		2.01		2.01	

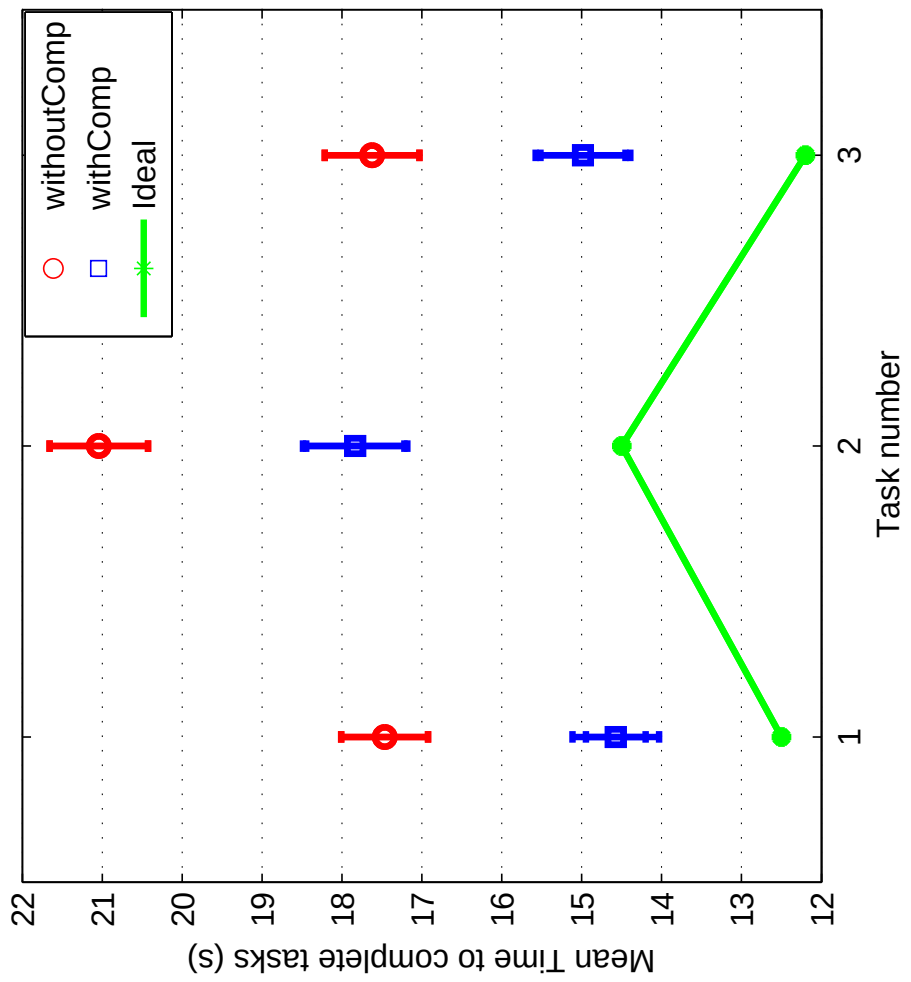

Figure 23: Mean TTCT in the good link condition with a confidence level of 95%.

Table 8: t-Test in the medium link conditions

	$\mathbf{TTCT}_{1_{med\_u}}$	$\mathbf{TTCT}_{1_{med\_c}}$	$\mathbf{TTC}_{2_{med\_u}}$	$\mathbf{TTCT}_{2_{med\_c}}$	$\mathbf{TTCT}_{3_{med\_u}}$	$\mathbf{TTCT}_{3_{med\_c}}$
Mean	29.36	21.11	32.72	23.95	27.78	19.67
Var	2.16	5.69	7.69	4.26	5.23	4.16
t Stat	15.01		12.92		13.50	
$p(T <= t)$ one-tail	8.8E-19		3.2E-17		1.95E-18	
t Critical one-tail	1.68		1.68		1.68	
$p(T <= t)$ two-tail	1.8E-18		6.4E-17		3.9E-18	
t Critical two-tail	2.02		2.01		2.01	

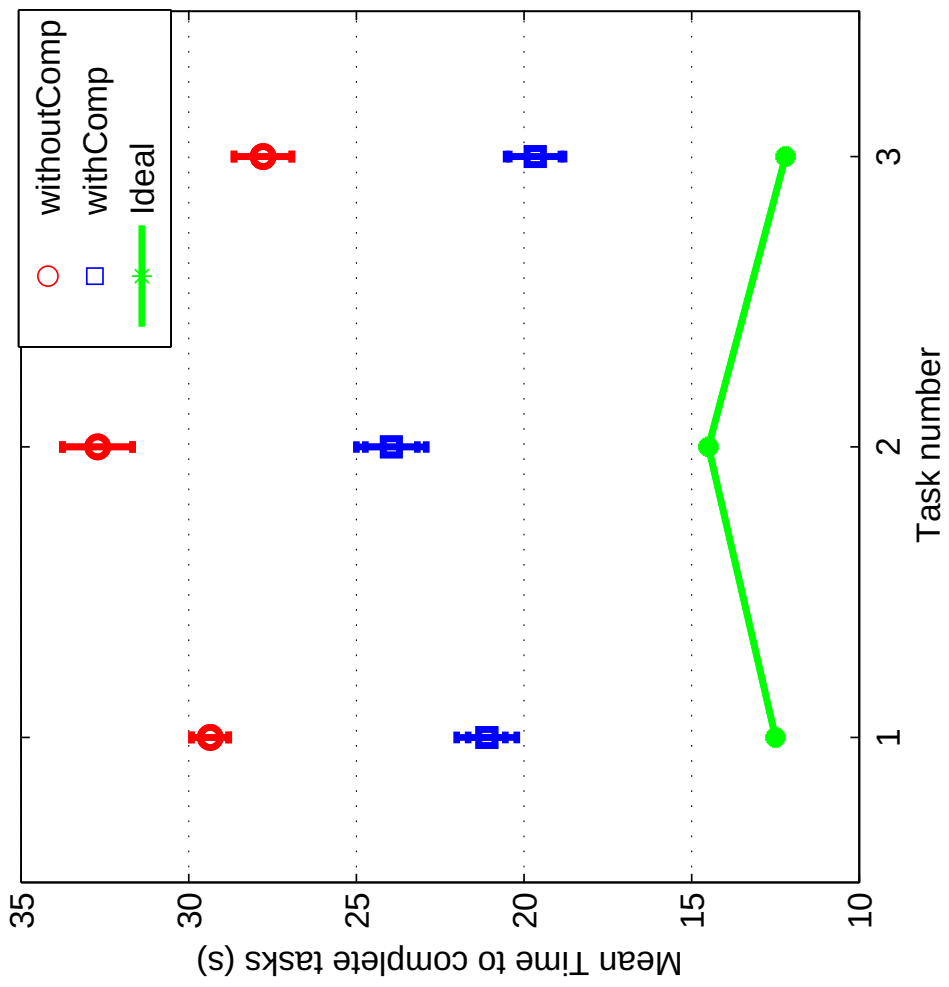

Figure 24: Mean TTCT in the medium link condition with a confidence level of 95%.

Table 9: t-Test in the bad link conditions

	$\mathbf{TTCT}_{1_{bad\_u}}$	$\mathbf{TTCT}_{1_{bad\_c}}$	$\mathbf{TTCT}_{2_{bad\_u}}$	$\mathbf{TTCT}_{2_{bad\_c}}$	$\mathbf{TTCT}_{3_{bad\_u}}$	$\mathbf{TTCT}_{3_{bad\_c}}$
Mean	48.27	31.40	55.96	36.23	49.52	30.32
Var	7.63	9.00	9.62	16.3	9.82	3.67
t Stat	21.09		19.76		26.65	
$p(T <= t)$ one-tail	7.8E-27		1.1E-24		8.2E-28	
t Critical one-tail	1.68		1.68		1.68	
$p(T <= t)$ two-tail	1.6E-26		2.1E-24		1.6E-27	
t Critical two-tail	2.01		2.01		2.02	

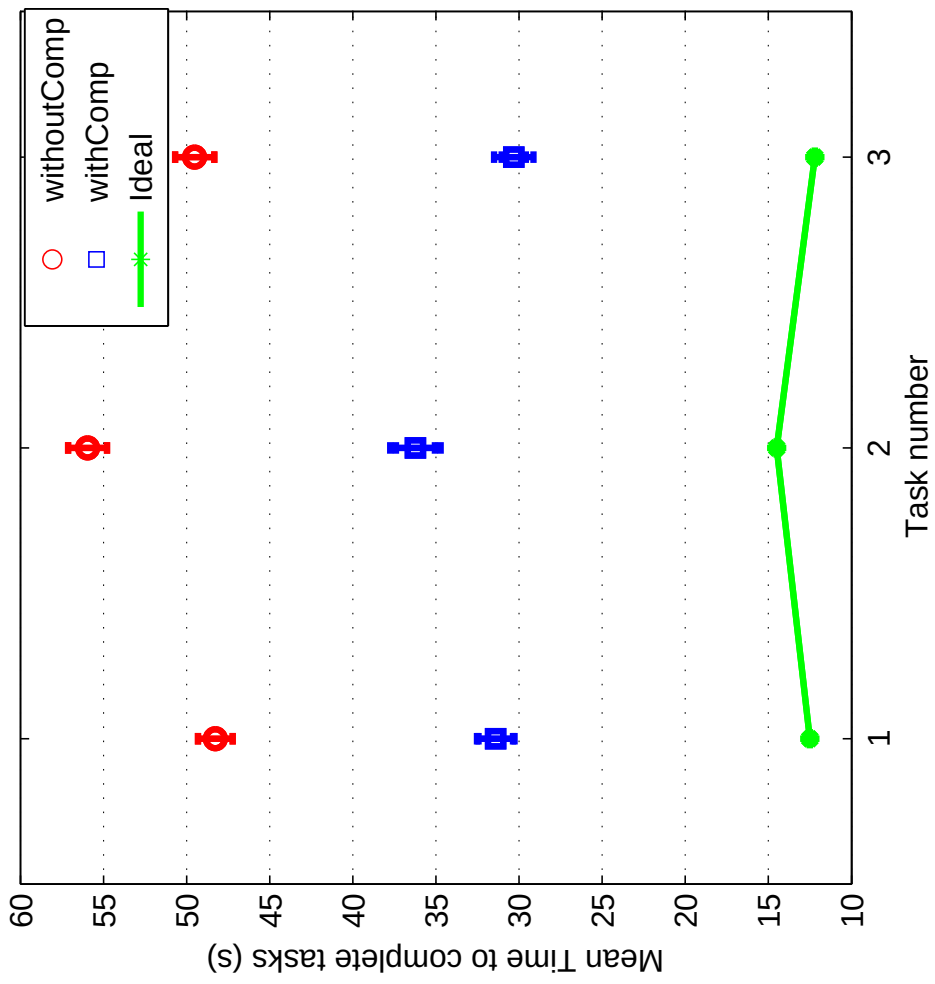

Figure 25: Mean TTCT in bad link condition with a confidence level of 95%.

Table 10: t-Test in the worst link conditions

	$\mathbf{TTCT}_{1_{wrs\_u}}$	$\mathbf{TTCT}_{1_{wrs\_c}}$	$\mathbf{TTCT}_{2_{wrs\_u}}$	$\mathbf{TTCT}_{2_{wrs\_c}}$	$\mathbf{TTCT}_{3_{wrs\_u}}$	$\mathbf{TTCT}_{3_{wrs\_c}}$
Mean	380	30.18	424.42	37.26	366.5	40.27
Var	28	22.29	20.97	23.62	56.74	3.69
t Stat	251.53		295.63		213.99	
$p(T <= t)$ one-tail	3.4E-78		4.8E-83		7.6E-47	
t Critical one-tail	1.68		1.68		1.70	
$p(T <= t)$ two-tail	6.7E-78		9.6E-83		1.5E-46	
t Critical two-tail	2.01		2.01		2.00	

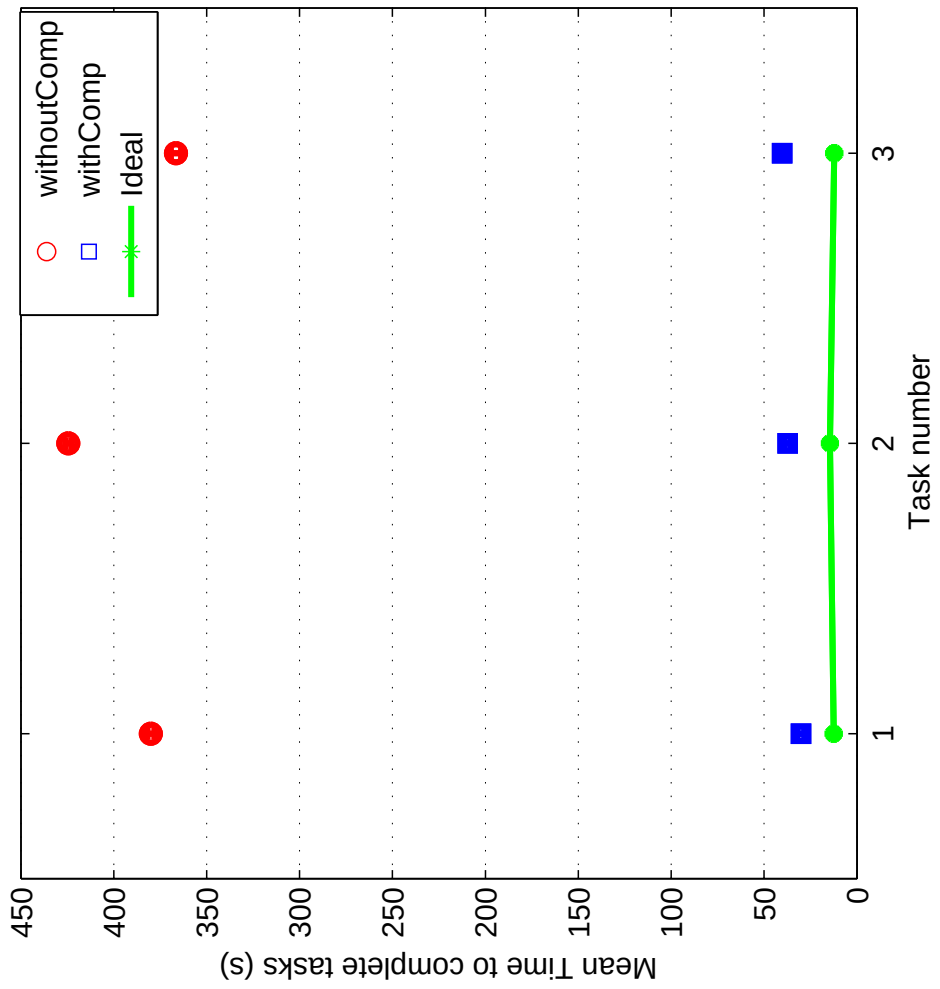

Figure 26: Mean TTCT in the worst link condition with a confidence level of 95%.

In summary, the t-tests show the statistical significance of the contribution of the compensation mechanism in good, medium, bad and worse conditions with the initial arrangement of values of ε_e as earlier depicted in Fig. 16. However, in order to study the impact of different combinations of ε_e on TTCT, a second phase of experiment was conducted with another set of 30 participants. Only the good and the bad channel conditions were considered in the experiment. Different sets of ε_e were chosen as shown in Table 11, when $0 < \varphi \leqslant 3$, $3 < \varphi \leqslant 5$, $5 < \varphi \leqslant 10$ and $\varphi > 10$, respectively. Notice in Table 11 that compensation set 1 was already used in the first phase of experiment.

Table 11: Different sets of mappings between φ and ε_e.

ε_e **set**	$\varphi\epsilon$**[0, 3]**	$\varphi\epsilon$**[3, 5]**	$\varphi\epsilon$**[5, 10]**	$\varphi\epsilon$**]10,** ∞**[**
1	300	700	1000	1500
2	1000	1000	700	300
3	1500	700	300	1
4	1500	700	700	300
5	1500	1000	300	300

Fig. 27 and 28 depict the TTCT for each set of ε_e (Table 11) when the channel is good and bad respectively. Notice that no significant change is observed when the channel is good as the confidence intervals overlap in all the test scenarios (see Fig. 27). However, Fig. 28 shows that compensation set five leads to a significant decrease in the TTCT for all the tasks completions because it employs longer compensation times for low values of φ.

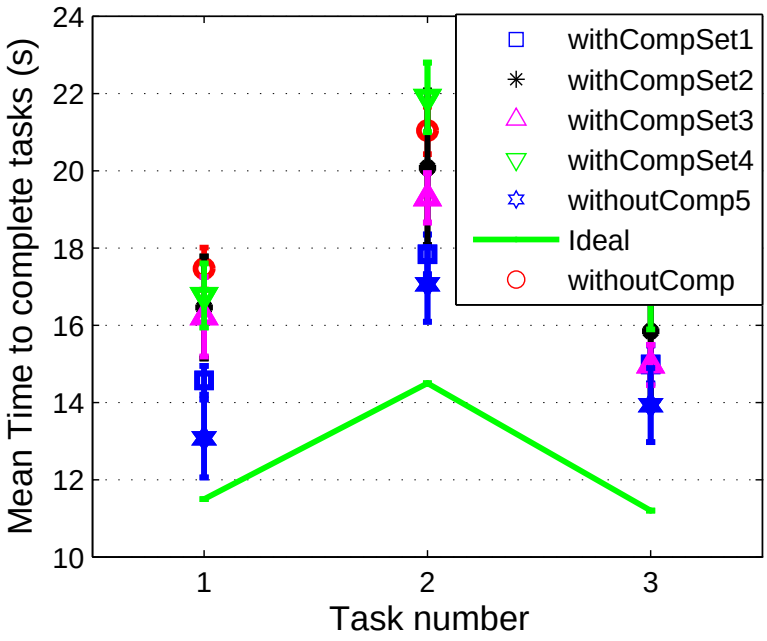

Figure 27: Mean TTCT in good channel condition with a confidence level of 95%.

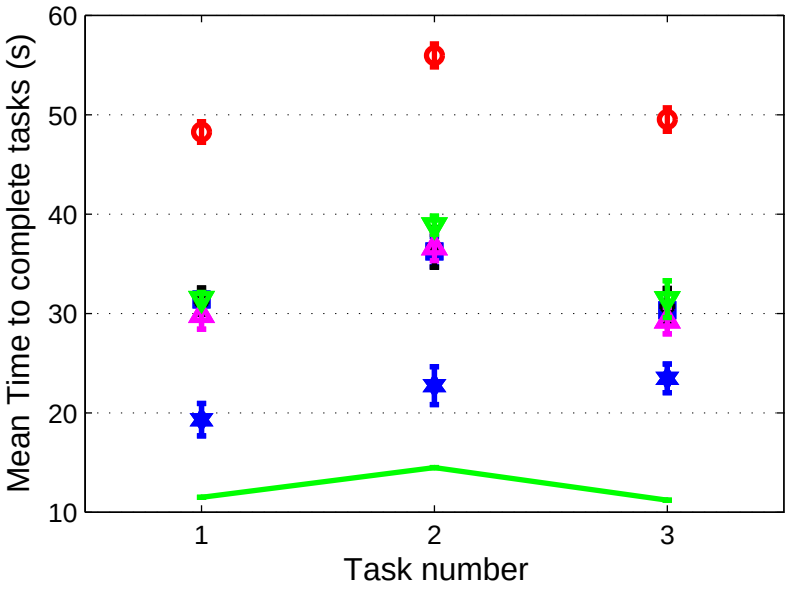

Figure 28: Mean TTCT in bad channel condition with a confidence level of 95%.

The efficiency compensation mechanism can be noticed from the increase in the gap between the blue squares and the red circles (see Fig. 23 through 26 as link condition degrades. These gaps can be translated into a percentage decrease in TTCT for tasks 1, 2, 3 as shown in Table 12. Notice in the table that the compensation mechanism has contributed in reducing approximately 16% of the TTCT in a good channel condition, while this percentage has increased to approximately 90% as the channel got worse.

Table 12: Percentage (%) decrease in TTCT due to compensation with respect to tasks and link conditions.

Link Condition	**Task 1**	**Task 2**	**Task 3**
Good	16.56	15.24	14.97
Medium	28.10	26.78	29.21
Bad	34.77	35.25	38.77
Worse	92.05	91.22	89.01

Fig. 29 depicts the evolution of the percentage decrease in TTCT which is a function of the degradation in link conditions. The link is initially in perfect condition (with 100% of received packets). As packet loss occurs the percentage of received packets decreases. When the percentage of packets lost gradually increases, the link condition degrades from perfect to very good, good, medium, bad and worse, respectively.

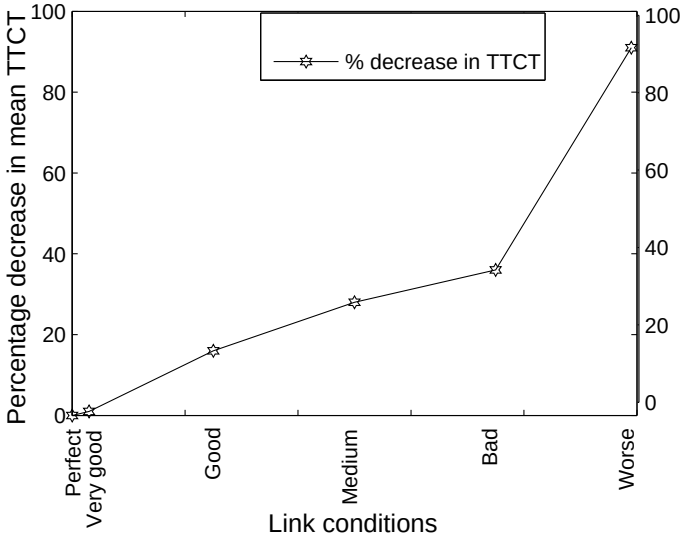

Figure 29: Trend of percentage decrease in TTCT

The black solid curve depicts the percentage decrease in the mean TTCT. Notice that the black solid curve in Fig. 29 shows that for each task completion the efficiency of the compensation algorithm in reducing the mean TTCT gradually increases as the link deteriorates. To illustrate this, points of lost percentages of 0, 3, 25, 50, 75 and 95. These values belong respectively to the range of lost percentage defining the perfect, very good, good, medium, bad and worst link conditions.

a. In very good link conditions, by assuming that 97% of the packets are successfully delivered to the CPD, the compensation mechanism does not have a great impact on the mean TTCT since it is seldom used by the system.

b. In good link conditions, by assuming that the percentage of received packets is 75%, compensation is performed for 25% of the total number of packets. This helps reduce the mean TTCT by approximately 16%.

c. In medium link condition, if the percentage of received packets is 50%, the system compensates for 50% of the total number of packets. This leads to a reduction in the mean TTCT by approximately 28%.

d. In bad link conditions, if the percentage of received packets is approximately 25%,

compensation is performed for packet losses of approximately 75%. This leads to a decrease in the mean TTCT by approximately 40%.

e. In worse link conditions, if the percentage of received packets is 1%, the compensation mechanism is expected to compensate for 99% of the total number of packets. This leads to more than 90% decrease in the mean TTCT. In worse link conditions, compensation is needed more often that in better link conditions. This explains the maximum decrease in the mean TTCT as shows the solid black curve in Fig. 29

Chapter 5: Conclusions

5.1 Summary

A compensation mechanism was designed, implemented and evaluated. Heuristics were used to set the compensation time ε_e. A one group pretest-postest scenario revealed that the compensation mechanism had significantly reduced the mean TTCT when the link was in good, medium, bad and worse conditions. A t-Test was run to confirm this statement and it was found that $p(T <= t)$ was less than 0.05. Therefore the null hypothesis which stated that *" no compensation can be achieved for QoE degradation under lossy wireless link conditions"* was rejected [3].

5.2 Limitations

Some limitations are identified in this research as discussed below:

a. The system could only remember what the state was at time $(T-1)$, but not a longer sequence of previous states.

b. The percentage of packet loss can only be computed when a packet is received. Hence, the system does not provide knowledge of the link condition prior to at least one packet arrival.

c. Impairment might have affected users' adaptation to the system. In addition, the state of mind of users could have affected the system evaluation. For instance, Sickness, tiredness, eye impairment, hunger, mood, etc. could have affected the user's ability to properly operate the CPD.

d. A single CPD was use throughout the experiment. So any unidentified issues associated with the CPD itself might have influenced the outcome.

e. At the maximum speed of three meters per second and in worse link conditions, the CPD might not promptly execute the operator's instructions for sudden changes in direction because the vehicle might switch to compensation mode for a longer period while new packets from the user might not be readily available to the CPD.

Future work might use pattern recognition in order to enable the system to remember longer sequences for performing compensation and also to enable finer control of the CPD.

References

[1] Andrew, N.(Instructor).(2015).*Machine learning* [Video lectures].
Retrieve from https://www.coursera.org/course/ml

[2] Bualat, M., et al. *Preparing for Crew-Control of Surface Robots from Orbit.*
Retrieved from https://www-preview.ri.cmu.edu/pub_files/iaa14-bualat-et-al.pdf

[3] Burke, J., & Larry, C. (2008). *Educational Research* (3rd ed.). Los Angeles, CA: Sage Publication, Inc.

[4] Jeff, L.(Instructor).(2015). *Practical machine learning* [Video lectures].
Retrieve from https://www.coursera.org/course/predmachlearn

[5] Julio, A.(2014). The Moving IoT. *Journal of Future Internet of Things and Cloud (FiCloud),*264-271.doi:10.1109/FiCloud.2014.49.

[6] Junaid, S., & Markus, F., & Denis, C. (2010). Quality of Experience from user and Network perspectives.*Ann Telecommun*, 65, 47-57.doi:DOI 10.1007/s12243-009-0142-x.

[7] Kevin, R. F., & Stevens, R.(2011).*TCP/IP Illustrated: Vol 1. The Protocols*(2nd ed.). Ann Arbor, MI: Edwards Brothers.

[8] Leonard, S.B. (1996). Basic elements of law. In M.E. Van (ed.), *Fundamentals of electrical engineering* (pp.3-24). New York, NY: Oxford University Press.

[9] MacKenzieand IS, Buxton W. (1992). Extending Fitts'law to two- dimensional tasks. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ser. CHI '92. New York, NY, USA: ACM. p. 219–226.

[10] Nordic Semiconductor.(2008). *nRF24l01+: Single chip 2.4GHz transceiver. Product specification v1.0.*
Retrieved from https://www.sparkfun.com/datasheets/Components/SMD/
nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf

[11] Parastoo S, Rodney AK, Predrag BR, Ramtin S.(2008). Finite-state Markov Modeling of fading channels. *IEEE signal processing magazine.*

[12] Schiele, A.(2011).*METERON Validating Orbit-to-Ground Telerobotics Operations Technologies,* 11^{th} *symposium on Advance Space Technologies in Robotics and Automation (ASTRA).*

Retrieved from http://robotics.estec.esa.int/ASTRA/Astra2011/Presentations /Plenary%202/03_schiele.pdf

[13] Sebastian T., & Wolfram B., & Dieter F. (2006). *Probabilistic Robotics* (pp.9-32). Cambridge, MA: The MIT press.

[14] Thomas, K.,& Schiele, A.(2013). *Exoskeleton control of the robonaut through rapid and ros.*

Retrieved from http://robotics.estec.esa.int/ASTRA/Astra2013/Papers/ Krueger_Schiele_2810861.pdf

Appendix A: Link Emulator

A.1 Statistical Summary of the Channel Characterization

Tables 13 and 14 present a statistical summary of the wireless channel characterization prior to removal of the outliers. This data was used to characterize the two state Markov model under the assumption of exponentially distributed state durationsl In the implementation conventional random variate generation mechanisms were employed [11].

Table 13: Statistical summary for the good state

Parameters	**5m**	**10m**	**15m**	**20m**
Mean	181762.2	495.295	272.667	51.517
Median	181762.2	22	52.8	26.4
Mode	NA	13.2	8.8	8.8
Std Deviation	40008.95	9766.267	1057.618	79.315
Skewness	NA	26.916	20.03	5.643
Range	56581.2	267097.6	30716.4	1311.2
Minimum	153471.6	8.8	8.8	8.8
Maximum	210052.8	267106.4	30725.2	1320
Sum	363524.4	376424.4	361829.6	112978.8
Count	2	760	1327	2193

Table 14: Statistical summary for the bad state

Parameters	**5m**	**10m**	**15m**	**20m**
Mean	7.2	36.942	41.936	84.676
Median	7.2	8.8	8.8	8.8
Mode	NA	4.4	4.4	4.4
Std Deviation	5.09	108.02	374.06	311.42
Skewness	NA	21.708	10.889	23.242
Range	7.2	2455.2	10700.8	13230.8
Minimum	3.6	4.4	4.4	4.4
Maximum	10.8	2459.6	10705.2	13235.2
Sum	14.4	78170.4	63575.6	327021.2
Count	2	2116	1516	3862

Appendix B: Description of the Hardware and the Mode of Operation

B.1 Controller

The controller unit comprises a joystick, two electronic boards and a wireless sensor (see Fig. 7). The relationship among these components is depicted in details in circuit diagram shown in Fig. 31. The joystick is the part of the controller which enables the user to perform the maneuvers to send command to the CPD. Its frame is made of a metallic outer core connected to a DC potential of 5 volts (supplied by master board A) and four independently isolated inner metallic plates (i.e. A, B, C, D) connected to digital the pins (i.e. D4, D7, D8 and D12) on master A. The inner plates are symmetrically arranged in a square pattern as shown in Fig. 30. At rest, the stick remains in the center of the square. So the outer core and inner plates are independently initially isolated.

Whenever a user wants to move the CPD in any direction (i.e. F, B, FR, FL, BR, BL, R or L) the bottom of the stick has to be moved towards a corresponding position (P1, P2, P3, P4, P5, P6, P7 or P8) as indicated in Fig. 30. When that happens, at least a switch SW={SA, SB, SC, SD} will close. But it is important to mention that when the stick moves to P1, P3, P5 or P7, only switches with similar switches close. On the other hand when the stick moves to P2, P4, P6 or P8, two different switches closed. In either case, when a switch closes a contact is established between the outer core (i.e.the core already connected to 5V) and whatever inner plate is connected to that switch. This enables current to flow from the outer core down to the inner metal plate through to the closed switch. Consequently any pin on master A that is connected to the inner plate will also be connected to 5V [8].

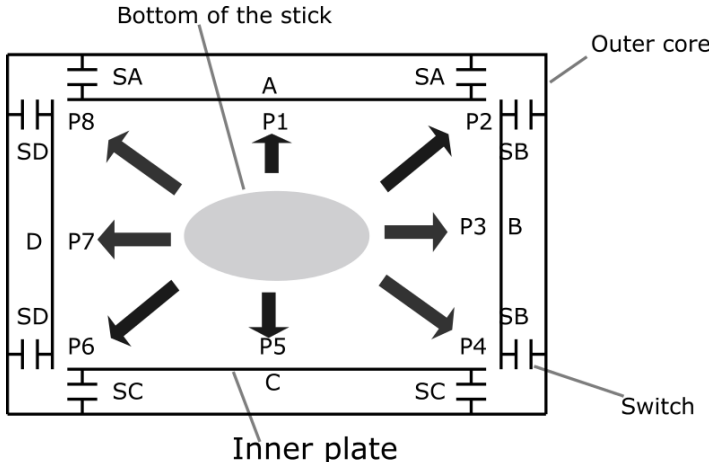

Figure 30: Schematic of the joystick

As earlier stated master A is set to interact with the joystick through its digital pins D4, D7, D8 and D12 (see Fig. 31). When a switch at the joystick closes, a 5 volt-analog signal flows through the switch down to a corresponding pin. Only in that case can the signal be interpreted by the microcontroller located on master A. Notice that each pin connected to an inner plate is also attached by default to ground through a one kilo-ohm (1KΩ) resistor for two reasons. First the resistor prevents the 5V signal (flowing from the joystick to the digital pins) from going to the ground. Second, the potential at the digital pins become zero when no 5V signal arrives from the joystick [8].

The signal interpretation by master A leads to generation of packets destined to be executed by the CPD. Packets are then sent to another electronic board which directly interacts with master A. Such a board is known as slave A. Packet generation is performed on the basis of an intermittent sampling. To explain, when a switch at the joystick is closed, master A initiates a clock which reset to zero after every 50 milliseconds. Within that time-frame master A checks whether the switch is really closed before it executes the corresponding instruction. Such a strategy is meant to avoid unintended packet generation by master A [5].

Slave A is controlled by master A (see Fig. 31). It receives packets from the latter through a "two-wire" interface (TWI). TWI is a communication protocol that governs data transfer via a serial data line (SDA) and a serial clock line (SCL) between master A and slave A.

Upon reception of packets, slave forwards them to transceiver A through a serial peripheral interface (SPI), which is another form of serial communication. After this stage, all packets are re-sized respect to the format supported by transceiver A and sensor B [10]. Such a format is referred to as Enhanced ShockBurst format where each packet comprises a 1 byte preamble, 5 byte-address, 9 bit-packet control field, 23 byte-payload and 2 byte-cyclic redundancy check (CRC). For clarifications on the pin arrangement for SPI protocol table 15 is provided, with a pin description of the wireless sensors made as follows.

Table 15: SPI pin arrangement slave A board-transceiver A

Electronic board	**Wireless transceiver**
GND	1.GND
3.3v	2.VCC
D9	3.CE
D10	4.CSN
D13	5.SCK
D11	6.MOSI
D12	7.MISO

- VCC pin powers the sensor with a voltage of 3.3V supplied by slave A.
- GND pin connects the sensor to ground.
- The "chip enable" (CE) pin is pulled high or low when the sensor is set to active mode; this mode means that the sensor is ready to transmit or receive data.
- The "chip select not"(CSN) pin is normally active when there is no voltage across it.

But when slave (A) wants to communicate with the sensor, it must be disabled. In other words the voltage across CSN must be pulled high (i.e. a logic 1).

- The clock (CSK) is decided by the electronic board directly connected to the sensor(i.e. slave A or master B in the present case).
- The "master in slave out" (MISO) pin is used by the sensor to respond to slave A through pin D12. However, no feedback is needed from transceiver B in this thesis as the communication had to simplex (i.e. without packet acknowledgement).
- The "master out slave in" (MOSI) pin and D11 on slave A form the data line used to send instructions to the sensor.

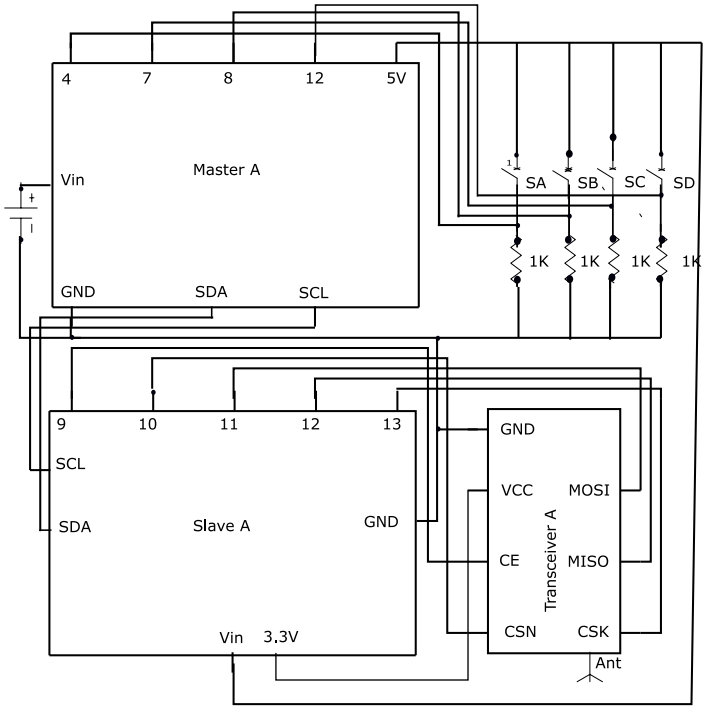

Figure 31: Circuit diagram of the controller

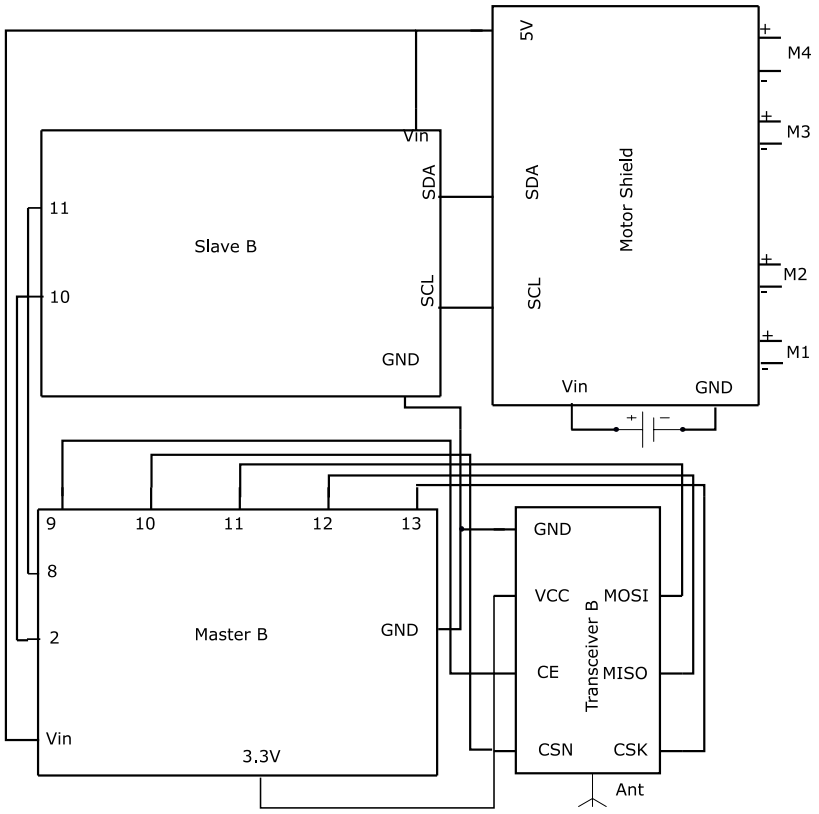

Figure 32: Circuit diagram of the CPD

B.2 CPD

B.2.1 Major Components and Interfaces

The wireless communication between transceiver A and transceiver B happened over the industrial, scientific and medical (ISM) radio frequency band at 2.4/2.5 MHz. To ensure delivery of packets, transceiver A is set to *transmit* mode while transceiver B (located on the CPD) is set to *receive* mode. Transceiver B continuously scans the radio channel) in order to detect packets when they arrives. For a packet to be successfully received by sensor B, the power of the frequency carrier must be at least -64 dBm [10].

Next, master B is tasked to retrieve packets from sensor B and forward them to slave B through a soft serial interface (see Fig. 32). Slave B performed two tasks. It is not only required to retrieve packets and packet loss information form master B but also compensate for packet losses whenever necessary. In addition it sends packets to the motor shield through TWI.

Finally, the motor controller reads the instructions and operates in turn the DC motors. Since decisions with regards to compensation for packet loss take place at slave B, it makes the final decision with regards to how the CPD should operate. Slave B may decide to simply forward commands as instructed by master B or add compensation instructions occasionally.

B.2.2 Tentative Trajectories of the CPD

The possible types of motions of the CPD are depicted in Fig. 33 and categorized into *nine*(9). These are identified next:

- (NM) represents "*no motion*" or "*idle position*". It reports the device at rest.
- (F) means "$forward$". In this motion CPD is expected to move forward only.
- (B) represents "*backward*". The CPD is expected to move backward only.
- (FR) is referred to as "$forward\ right$". In this mode the vehicle goes forward while slightly deviating to the right.
- (FL) represents "$forward\ left''$. The CPD moves forward while slightly deviating to the left.
- (BR) is referred to as "*backward right*". In this mode the vehicle moves backward while slightly deviating to the right.
- (BL) stands for "*backward* $left$". The CPD is expected to move backward while slightly deviating to the left.
- (R) symbolizes "$right$" to indicate a sharp turn on the right or a clockwise rotation.
- (L) represents "$left$" to indicate a sharp turn on the left or an anticlockwise rotation.

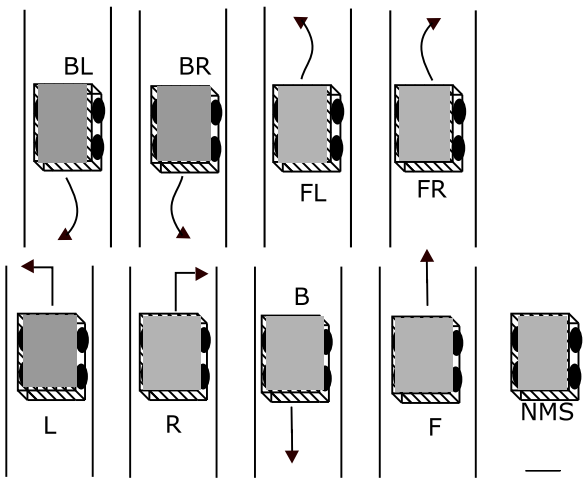

Figure 33: Possible types of motions for the CPD

Appendix C: Description of the Software

This section discusses the scripts created to enable interaction among the various units involved. Instructions are entirely written in "C" language. In total five programs are written with each implemented on a separate board as details below. Two sets of codes run at the controller. One is embedded on master A to instruct how commands should be generated and another is implemented on slave A to tells how to link emulation and packet counting should be done. The CPD is driven by three sets of programs. The script on master B enables packet reception and processing, while slave B runs the compensation and command forwarding program. Finally, the motor shield relies on another script to communicate with the motors. Depending on the type of interface a the functionality different libraries are used with these programs.

C.1 Command Generation

The following program specifies the interface used to ensure communication between master A and slave A. It also describes how the analog signal resulting from manipulation of the joystick is converted into numerical 8 bit packets. Moreover, it clarifies how often the microcontroller checked the status (i.e. high or low) of the pins connected to the joystick.

```
#include <Wire.h> //enable two wire interface
int pin12 = 12; //pin declaration
int led13 = 13;
int pin4 = 4;
int pin7 = 7;
int pin8 = 8;
unsigned long f=2; //command definition
unsigned long fl=3;
unsigned long fr=5;
unsigned long lt=6;
unsigned long rt=4;
unsigned long b=8;
```

```
unsigned long bl=7;
unsigned long br=9;
unsigned long s=0;
long lastDebounceTime = 0; //timing parameters
long debounceDelay = 30;
int stopTime=1000;
unsigned long initialMillis=0;
void setup(){
pinMode(pin12, INPUT\_PULLUP);
pinMode(pin4, INPUT\_PULLUP);
pinMode(pin7, INPUT\_PULLUP);
pinMode(pin8, INPUT\_PULLUP);
pinMode(led13, OUTPUT);
digitalWrite(led13, LOW);
Serial.begin(9600);
Wire.begin();
}
void loop(){
int reading12 = digitalRead(pin12);
int reading4 = digitalRead(pin4);
int reading7 = digitalRead(pin7);
int reading8 = digitalRead(pin8);
if (((reading12 == HIGH)&&(reading4 == LOW))&&
((reading7 == LOW)&&(reading8 == LOW)))
{lastDebounceTime = millis();
if ((millis() - lastDebounceTime) < debounceDelay)
{
digitalWrite(led13, HIGH);
Wire.beginTransmission(5);
```

```
Wire.write(f);
Wire.endTransmission();
initialMillis = millis();
}
else if ((millis() - lastDebounceTime) >
debounceDelay)
{
digitalWrite(led13, LOW);
}
else if (((reading12 == LOW)&&(reading4 == HIGH)&&
((reading7 == LOW)&&(reading8 == LOW)))
{{
lastDebounceTime = millis();
}
if ((millis() - lastDebounceTime) < debounceDelay)
{
digitalWrite(led13, HIGH);
Wire.beginTransmission(5);
Wire.write(lt);
Wire.endTransmission();
initialMillis = millis();
}
else if ((millis() - lastDebounceTime) > debounceDelay)
{
digitalWrite(led13, LOW);
}}
else if (((reading12 == LOW)&&(reading4 == LOW))
&&((reading7 == HIGH)&&(reading8 == LOW)))
{{
```

```
lastDebounceTime = millis();
}
if ((millis() - lastDebounceTime) $<$ debounceDelay)
{
digitalWrite(led13, HIGH);
Wire.beginTransmission(5);
Wire.write(b);
Wire.endTransmission();
initialMillis = millis();
}
else if ((millis() - lastDebounceTime) > debounceDelay)
{
digitalWrite(led13, LOW);
}}
else if (((reading12 == LOW)&&(reading4 == LOW))&&
((reading7 == LOW)&&(reading8 == HIGH)))
{{
lastDebounceTime = millis();
}
if ((millis() - lastDebounceTime) < debounceDelay)
{
digitalWrite(led13, HIGH);
Wire.beginTransmission(5);
Wire.write(rt);
Wire.endTransmission();
initialMillis = millis();
}
else if ((millis() - lastDebounceTime) > debounceDelay)
{
```

```
digitalWrite(led13, LOW);
}}
else if (((reading12 == HIGH)&&(reading4 == LOW))&&
((reading7 == LOW)&&(reading8 == HIGH)))
{
lastDebounceTime = millis();
if ((millis() - lastDebounceTime) < debounceDelay)
{
digitalWrite(led13, HIGH);
Wire.beginTransmission(5);
Wire.write(f1);
Wire.endTransmission();
initialMillis = millis();
}
else if ((millis() - lastDebounceTime) > debounceDelay)
{
digitalWrite(led13, LOW);
}}
else if (((reading12 == HIGH)&&(reading4 == HIGH))&&
((reading7 == LOW)&&(reading8 == LOW)))
{{
lastDebounceTime = millis();
}
if ((millis() - lastDebounceTime) < debounceDelay)
{
digitalWrite(led13, HIGH);

Wire.beginTransmission(5);
```

```
Wire.write(fr);

Wire.endTransmission();

initialMillis = millis();
}
else if ((millis() - lastDebounceTime) > debounceDelay)
{
digitalWrite(led13, LOW);
}}
else if (((reading12 == LOW)&&(reading4 == HIGH))&&
((reading7 == HIGH)&&(reading8 == LOW)))
{
lastDebounceTime = millis();
if ((millis() - lastDebounceTime) < debounceDelay)
{
digitalWrite(led13, HIGH);
Wire.beginTransmission(5);
Wire.write(br);
Wire.endTransmission();
initialMillis = millis();
}
else if ((millis() - lastDebounceTime) > debounceDelay)
{
digitalWrite(led13, LOW);
}}
else if (((reading12 == LOW)&&(reading4 == LOW))&&
((reading7 == HIGH)&&(reading8 == HIGH)))
{{
```

```
lastDebounceTime = millis();
}
if ((millis() - lastDebounceTime) < debounceDelay)
{
digitalWrite(led13, HIGH);
Wire.beginTransmission(5);
Wire.write(bl);
Wire.endTransmission();
}
else if ((millis() - lastDebounceTime) > debounceDelay)
{
digitalWrite(led13, LOW);
}}
else if (((reading12 == LOW)&&(reading4 == LOW))&&
((reading7 == LOW)&&(reading8 == LOW)))
{{
lastDebounceTime = millis();
}
if ((millis() - lastDebounceTime) < debounceDelay)
{
if (unsigned long(millis() - initialMillis) <= stopTime)
{
for(i=0; i<60;i++){
digitalWrite(led13, HIGH);
Wire.beginTransmission(5);
Wire.write(s);
Wire.endTransmission();
Serial.println(``s");
}}
```

```
else if ((millis() - lastDebounceTime) > debounceDelay)
{
digitalWrite(led13 , LOW);
}}}
}}
```

C.2 Link Emulation and Packet Counter Initiation

This code mainly performs the following operations. First, it instructs slave A be to listen to master A over the TWI interface in order to receive the generated commands. In addition, it specifies that communication between slave A and transceiver A should happen over SPI. Furthermore, this program enables slave A to generate and insert consecutive numbers into consecutive packets, respectively. It also provides an adjustable link emulation technique to simulate link degradation. Finally, it instructs slave A to convert each packet payload form 8 bit-format to 32bytes. Again, decisions with regards to the size of the CRC, the data rate, the suppression of the acknowledgement, and the selection of the transmission channel are made here.

```
#include <SPI.h>
#include ``nRF24L01.h"
#include ``RF24.h"
#include ``printf.h"
#include <Wire.h>
unsigned long count=10; packet counter set
RF24 radio(9,10); CE & CSN pins initialized
const uint64_t pipes[2] = {0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL};
typedef enum {role_ping_out = 1, role_pong_back } role_e;
const char* role_friendly_name[]={``invalid",
``Ping out", ``Pong back"};
role_e role = role_ping_out;
float randNumber;
int x=10000;
```

```
int a=99; //mean duration
// of the good state ranging from 1 to 99
int b=1; //mean duration of the bad state
// ranging from 1 to 99
float goodStateDuration; // timing parameters
float badStateDuration;
long goodStateStart;
long badStateStart;
char Good= 'G';
char Bad= 'B';
void receiveEvent (int howMany)
{}
bool enable=false;
void setAutoAck(bool enable);//auto acknowledgement disabled
void setup(void)
{
Wire.begin(5);
Wire.onReceive(receiveEvent);
Serial.begin(57600);
printf_begin();
printf(''ROLE: %s\n\r",role_friendly_name[role]);
radio.begin();// radio interface initiated
radio.setDataRate(RF24_71KPS); // transmit at a data
// rate of 2 Mbps
radio.setCRCLength(RF24_CRC_16); // 16 bit cyclic
//redundancy check enabled
radio.setPayloadSize(23); //payload size set
//to 23 bytes
role = role_ping_out;
```

```
radio.openWritingPipe(pipes[0]); // transmit channel enabled
radio.openReadingPipe(1,pipes[1]); // receiving channel
// enabled even though not necessary used here
radio.startListening(); //expecting incoming data
// even though not necessary here
radio.printDetails(); // some sensor parameters printed
}
void loop()
{
randNumber = random(1, 10000);
goodStateDuration = -a*log(randNumber/x);
goodStateStart = millis();
delay(1);
badStateDuration = -b*log(1-(randNumber/x));
if ((millis()- goodStateStart) < goodStateDuration)
{
Serial.print(goodStateDuration);
Serial.println(Good);
radio.stopListening();
if(Wire.available())
{
unsigned long packet = toupper(Wire.read());
bool ok = radio.write( &packet, sizeof(unsigned long) );
radio.write( &count, sizeof(unsigned long) );
count=count+1;
printf(``%lu\n",packet);
printf(``%lu\n",count);
radio.startListening();
}
```

```
else
{
count=10;
radio.stopListening();//transmission
}}
else if ((millis()- goodStateStart) > goodStateDuration)
{
Serial.print(badStateDuration);
Serial.println(badStateDuration);
count=count+(badStateDuration/3.6); // determine the number
//of packets that would have been lost
delay(badStateDuration); // remain silent over this period
}
}
```

C.3 Packet Reception and Processing

This code runs at the receiver side. It tells master B to retrieve packets from transceiver B through SPI. Packets are now dissociated form their attached numbers before computation of packet loss takes place. Then packets and loss information are forwarded to slave B via a softserial path. Noticed that acknowledgement is disabled. This code also provides specification with regards to other parameters such as data rate, CRC, channel and payload size to match the settings at transmitter.

```
#include <SPI.h>
#include ``nRF24L01.h"
#include ``RF24.h"
#include ``printf.h"
#include <SoftwareSerial.h>
SoftwareSerial mySerial(8, 2); //RX, TX
unsigned long oldPacket=9;
unsigned long n=0;
```

```
unsigned long loss1=0;
int interval=100;
int wait=5;
unsigned long previousMillis=0;
unsigned long initialMillis=0;
unsigned long percentLoss;
RF24 radio(9,10);
const uint64_t pipes[2] = {0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL};
typedef enum {role_ping_out = 1, role_pong_back} role_e;
const char* role_friendly_name[] = {"invalid",
 "Ping out", "Pong back"};
role_e role = role_pong_back;
bool enable = false;
void setAutoAck(bool enable);
void setup(void){
Serial.begin(57600);
mySerial.begin(4800);
printf\_begin();
printf("ROLE:%s\n\r",role_friendly_name[role]);
radio.begin();
radio.setDataRate(RF24_71KPS);
radio.setPayloadSize(23);
radio.openReadingPipe(1,pipes[1]);
radio.startListening();
radio.printDetails();
}
void loop(void){
role = role_pong_back;
radio.openWritingPipe(pipes[1]);
```

```
radio.openReadingPipe(1,pipes[0]);
if ( role == role_pong_back )
{
bool goodSignal = radio.testRPD();
if (radio.available()){Serial.println(goodSignal?
``Strong signal > -64dBm'':``Weak signal < -64dBm'');
radio.read(0,0);
printf(``RPD %d\n'',goodSignal);
unsigned long newPacket;
bool done = false;
while (!done)
{
unsigned long currentMillis = millis();
if (newPacket<10)
{
mySerial.write(newPacket);
}
else if (NewPacket>9)
{
unsigned long loss = newPacket-oldPacket-1;
if ((unsigned long)(currentMillis - initialMillis) >= wait)
{
if (newPacket != oldPacket)
{
loss1 = loss1+loss;
printf(``loss %lu\n'',loss);
printf(``Number \%lu\backslash$n'',newPacket);
n = n+1;
oldPacket =newPacket;
```

```
}}
digitalWrite(13,HIGH);
}
if ((unsigned long)(currentMillis - previousMillis) >= interval)
{
printf(got %lu\n",n);
printf(``Total loss %lu\n",loss1);
percentLoss= ((100*loss1)/(n+loss1));
printf(``percent loss %lu\n",percentLoss);
n=0;
loss1=0;
printf("total time %lu\n",(currentMillis - previousMillis));
previousMillis = currentMillis;
}
}
delay(1);
}}}
```

C.4 Codes for Slave B

C.4.1 Command Forwarding Only

The following code makes slave B behave neutrally. To explain, it enables slave B to only retrieve packets and forward them to the motor controller without taking any further course of action. Therefore, the behavior of the CPD directly results form the changing link conditions. This code helped build a baseline for the system study.

```
#include <Wire.h>
#include <SoftwareSerial.h>
unsigned long percentLoss;
SoftwareSerial mySerial(10, 11); //RX, TX
void setup(void)
{
```

```
Serial.begin(57600);
Wire.begin();
mySerial.begin(4800); // data rate set
//for the SoftwareSerial port
pinMode(13, OUTPUT);//LED
}
void loop(void)
{
while (mySerial.available())
{
unsigned long currentCommand = mySerial.read();
if (currentCommand == 1)
{
Wire.beginTransmission(5);
Wire.write(1);
Wire.endTransmission();
}
else if(currentCommand == 2)
{
Wire.beginTransmission(5);
Wire.write(2);
Wire.endTransmission();
}
else if(currentCommand == 3)
{
Wire.beginTransmission(5);
Wire.write(3);
Wire.endTransmission();
}
```

```
else if(currentCommand == 4)
{
Wire.beginTransmission(5);
Wire.write(4);
Wire.endTransmission();
}
else if(currentCommand == 5)
{
Wire.beginTransmission(5);
Wire.write(5);
Wire.endTransmission();
}
else if(currentCommand == 6)
{
Wire.beginTransmission(5);
Wire.write(6);
Wire.endTransmission();
}
else if(currentCommand == 7)
{
Wire.beginTransmission(5);
Wire.write(7);
Wire.endTransmission();
}
else if(currentCommand == 8)
{
Wire.beginTransmission(5);
Wire.write(8);
Wire.endTransmission();
```

```
}
else if(currentCommand == 9)
{
Wire.beginTransmission(5);
Wire.write(9);
Wire.endTransmission();
}
}
delay(2);
}
```

C.4.2 Initial Script for Compensation and Command Forwarding

This code enables slave B not only to forward packets received, but also make decisions depending on the link conditions. Four levels of compensation are provided to help the system keep track of the user's need and perform tasks accordingly. The traffic counter, control instructor and the idle controller are initiated here in order to enable slave B to refer to a previous state at time ($t = T - 1$) and initiate a compensation whenever loss occurs. In addition, this program accounts for compensation for extraneous packet delay. The I2C protocol is used to govern communication between slave B and the motor controller.

```
#include <Wire.h>
#include <SoftwareSerial.h>
unsigned long percentLoss;
SoftwareSerial mySerial(10, 11); // RX, TX
unsigned long currentCommand;
int compensationTime=1000;
int highCompensationTime=1500;
int lowCompensationTime=300;
unsigned long initialMillis=0;
unsigned long comp;
unsigned long startMillis=0;
```

```
unsigned long   waitMillis=0;
int i;
int n=0;
int k=0;
int m=0;
int t=0;
int s=0;
void setup(void)
{
Serial.begin(57600);
Wire.begin();
mySerial.begin(4800); //data rate set
//for the SoftwareSerial port
pinMode(13, OUTPUT);// LED
}
void loop(void)
{
while (mySerial.available())
{
unsigned long currentCommand = mySerial.read();
if (currentCommand > 9)
{
percentLoss = currentCommand - 10;
Serial.println(percentLoss);
t=t+1;
}
if(percentLoss == 0 && t>=300)
{
if (currentCommand == 2)
```

```
{
Wire.beginTransmission(5);
Wire.write(2);
Wire.endTransmission();
Serial.println(2);
comp=2;
m=m+1;
}
else if (currentCommand == 3)
{
Wire.beginTransmission(5);
Wire.write(3);
Wire.endTransmission();
Serial.println(3);
comp=3;
m=6;
}
else if (currentCommand == 4)
{
Wire.beginTransmission(5);
Wire.write(4);
Wire.endTransmission();
Serial.println(4);
comp=4;
m=6;
}
else if (currentCommand == 5)
{
Wire.beginTransmission(5);
```

```
Wire.write(5);
Wire.endTransmission();
Serial.println(5);
comp=5;
m=6;
}
else if (currentCommand == 6)
{
Wire.beginTransmission(5);
Wire.write(6);
Wire.endTransmission();
Serial.println(6);
comp=6;
m=6;
}
else if (currentCommand == 7)
{
Wire.beginTransmission(5);
Wire.write(7);
Wire.endTransmission();
Serial.println(7);
comp=7;
m=6;
}
else if (currentCommand == 8)
{
Wire.beginTransmission(5);
Wire.write(8);
Wire.endTransmission();
```

```
Serial.println(8);
comp=8;
m=m+1;
}
else if (currentCommand == 9)
{
Wire.beginTransmission(5);
Wire.write(9);
Wire.endTransmission();
Serial.println(9);
comp=9;
m=6;
}
else if (currentCommand == 0)
{
m=0;
n=n+1;
comp=0;
s=0;
}
if(m>5 && m<=10)
{k=1;}
else if(m>3 && m<=5)
{k=0;}
else if(m>10)
{k=2;}
else if(m<=3){k=10;}
}
else if(percentLoss==0&&t<300)
```

```
{
if (currentCommand == 2)
{
if (s<=20)
{
Serial.println(2);
comp=2;
t=0;
m=m+1;
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
Wire.write(2);
Wire.endTransmission();
Serial.println(``low level anticipation 2");
n=n+1;
if(n==10){n=20; comp=0;}
}}
else if (s>20)
{
Serial.println(2);
comp=2;
t=0;
m = m+1;
for(i=0; i<50;i++)
{
Wire.beginTransmission(5);
Wire.write(2);
Wire.endTransmission();
```

```
Serial.println(``anticipated 2");
n=n+1;
if(n==50){n=60; comp=0;}
}}}
else if (currentCommand == 3)
{
if (s<=20)
{
Serial.println(3);
comp=3;
t=0;
 m=m+1;
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
Wire.write(3);
Wire.endTransmission();
Serial.println ``low level anticipation 3");
n=n+1;
if(n==10){n=20; comp=0;}
}}
else if (s>20)
{
Serial.println(3);
comp=3;
t=0;
m=m+1;
for(i=0; i<50;i++)
{
```

```
Wire.beginTransmission(5);
Wire.write(3);
Wire.endTransmission();
Serial.println(``anticipated 3");
n=n+1;
if(n==50){n=60; comp=0;}
}}}
else if (currentCommand == 4)
{
if (s<=20)
{
Serial.println(4);
comp=4;
t=0;
m=m+1;
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
Wire.write(4);
Wire.endTransmission();
Serial.println(``low level anticipation 4");
n=n+1;
if(n==10){n=20; comp=0;}
}}
else if (s>20)
{
Serial.println(4);
comp=4;
t=0;
```

```
m=m+1;
for(i=0; i<50;i++)
{
Wire.beginTransmission(5);
Wire.endTransmission();
Serial.println(``anticipated 4");
n=n+1;
if(n==50){n=60; comp=0;}
}}}
else if (currentCommand == 5)
{
if (s<=20)
{
Serial.println(5);
comp=5;
t=0;
m=m+1;
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
Wire.write(5);
Wire.endTransmission();
Serial.println(``low level anticipation 5");
n=n+1;
if(n==10){n=20; comp=0;}
}}
else if (s>20)
{
Serial.println(5);
```

```
comp=5;
t=0;
m=m+1;
for(i=0; i<50;i++)
{
Wire.beginTransmission(5);
Wire.write(5);
Wire.endTransmission();
Serial.println(``anticipated 5");
n=n+1;
if(n==50){n=60; comp=0;}
}}}
else if (currentCommand == 6)
{
if (s<=20)
{
Serial.println(6);
comp=6;
t=0;
m=m+1;
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
Wire.write(6);
Wire.endTransmission();
Serial.println(``low level anticipation 6");
n=n+1;
if(n==10){n=20; comp=0;}
}
```

```
else if (s>20)
{
Serial.println(6);
comp=6;
t=0;
m=m+1;
for(i=0; i<50;i++)
{
Wire.beginTransmission(5);
Wire.write(6);
Wire.endTransmission();
Serial.println(``anticipated 6");
n=n+1;
if(n==50){n=60; comp=0;}
}}}
else if (currentCommand == 7)
{
if (s<=20)
{
Serial.println(7);
comp=7;
t=0;
m=m+1;
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
Wire.write(7);
Wire.endTransmission();
Serial.println(``low level  anticipation 7");
```

```
n=n+1;
if(n==10){n=20; comp=0;}
}}
else if (s>20)
{
Serial.println(7);
comp=7;
t=0;
m=m+1;
for(i=0; i<50;i++)
{
Wire.beginTransmission(5);
Wire.write(7);
Wire.endTransmission();
Serial.println(``anticipated 7");
n=n+1;
if(n==50){n=60; comp=0;}
}}}
else if (currentCommand == 8)
{
if (s<=20)
{
Serial.println(8);
comp=8;
t=0;
m=m+1;
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
```

```
Wire.write(8);
Wire.endTransmission();
Serial.println(``low level anticipation 8");
n=n+1;
if(n==10){n=20; comp = 0;}
Serial.println(``m");
}}
else if (s>20)
{
Serial.println(8);
comp=8;
t=0;
m=m+1;
for(i=0; i<50;i++)
{
Wire.beginTransmission(5);
Wire.write(8);
Wire.endTransmission();
Serial.println(``anticipated 8");
n=n+1;
if(n==50){n=60; comp=0;}
}}}
else if (currentCommand == 9)
{
if (s<=20)
{
Serial.println(9);
comp=9;
t=0;
```

```
m=m+1;
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
Wire.write(9);
Wire.endTransmission();
Serial.println(``low level anticipation 9");
n=n+1;
if(n==10){n=20; comp=0;}
}}
else if (s>20)
{
Serial.println(9);
comp=9;
t=0;
m=m+1;
for(i=0; i<50;i++)
{
Wire.beginTransmission(5);
Wire.write(9);
Wire.endTransmission();
Serial.println(``anticipated 9");
n=n+1;
if(n==50){n=60; comp=0;}
}}}
else if (currentCommand == 0)
{
m=0;
n=n+1;
```

```
comp=0;
s=0;
}
if(m>5 && m<=10){k=2;}
else if(m>3 && m<=5){k=1;}
else if(m>10){k=3;}
else if(m<=3){k=0;}
}}
if (!mySerial.available()&& percentLoss>0)
{
if (k==1)
{
Serial.println(``Slow comp");
n=1;
if (comp==2 && n<50)
{
for(i=0; i<500;i++)
{
Wire.beginTransmission(5);
Wire.write(2);
Wire.endTransmission();
Serial.println(``compensated 2");
n=n+1;
if(n==500){n=510; comp=0;}
Serial.println(n);
Serial.println(``m");
}
else if (comp==8&&n<50)
{
```

```
for(i=0; i<500;i++)
{
Wire.beginTransmission(5);
Wire.write(8);
Wire.endTransmission();
Serial.println(``compensated 8");
n=n+1;
if(n==500){n=510; comp=0;}
Serial.println(n);
}}
else if (comp==6 && n<50)
{
for(i=0; i<20;i++)
{
Wire.beginTransmission(5);
Wire.write(6);
Wire.endTransmission();
Serial.println(``compensated 6");
n=n+1;
if(n==50){=60; comp=0;}
Serial.println(n);
}}
else if (comp==4&&n<50)
{
for(i=0; i<20;i++)
{
Wire.beginTransmission(5);
Wire.write(4);
Wire.endTransmission();
```

```
Serial.println(``compensated 4");
n=n+1;
if(n==50){n=510; comp=0;}
Serial.println(n);
}}
else if (comp == 3 && n<50)
{
for(i=0; i<500;i++)
{
Wire.beginTransmission(5);
Wire.write(3);
Wire.endTransmission();
n=n+1;
if(n==500){n=510; comp=0;}
Serial.println(n);
Serial.println(``compensated 3");
}}
else if (comp == 7 && n<50)
{
for(i=0; i<500;i++)
{
Wire.beginTransmission(5);
Wire.write(7);
Wire.endTransmission();
Serial.println(``compensated 7");
n=n+1;
if(n==500){n=510; comp=0;}
Serial.println(n);
}}
```

```
else if (comp == 9 && n<50)
{
for (i=0; i <500; i++)
{
Wire.beginTransmission (5);
Wire.write (9);
Wire.endTransmission ();
Serial.println (``compensated 8");
n=n+1;
if(n==500){n=510; comp=0;}
Serial.println(n);
}}
else if (comp == 5 && n<50)
{
for(i=0; i <500; i++)
{
Wire.beginTransmission (5);
Wire.write (5);
Serial.println (``compensated 5");
n=n+1;
if(n==500){n=510; comp=0;}
Serial.println(n);
}}
else if (n>0 && comp==0)
{
percentLoss = 0;
n=1;
}}
else if (k==2)
```

```
{
Serial.println(``Level three compensation");
unsigned long currentMillis = millis();
if ((unsigned long)(currentMillis - initialMillis)>=
compensationTime)
{
percentLoss=0;
initialMillis = currentMillis;
}
if (comp == 2)
{
for(i=0; i<60;i++)
{
Wire.beginTransmission(5);
Wire.write(2);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if (comp == 8)
{
for(i=0; i<60;i++)
{
Wire.beginTransmission(5);
Wire.write(8);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if (comp == 6)
{
```

```
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
Wire.write(6);
Wire.endTransmission();
Serial.println(``compensated 6");
}}
else if (comp == 4)
{
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
Wire.write(4);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if (comp == 3)
{
for(i=0; i<60;i++)
{
Wire.beginTransmission(5);
Wire.write(3);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if (comp == 7)
{
for(i=0; i<60;i++)
{
```

```
Wire.beginTransmission(5);
Wire.write(7);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if (comp == 9)
{
for(i=0; i<60;i++)
{
Wire.beginTransmission(5);
Wire.write(9);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if(comp==5)
{
for(i=0;i<60;i++)
{
Wire.beginTransmission(5);
Wire.write(5);
Wire.endTransmission();
Serial.println(``compensated");
}}}
else if (k == 3)
{
Serial.println(``High level compensation");
unsigned long currentMillis = millis();
if(unsigned long(currentMillis - startMillis)>=
highCompensationTime)
```

```
{
percentLoss = 0;
startMillis = currentMillis;
}
if (comp == 2)
{
for (i=0; i<100; i++)
{
Wire.beginTransmission(5);
Wire.write(2);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if (comp == 8)
{
for(i=0; i<100; i++)
{
Wire.beginTransmission(5);
Wire.write(8);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if (comp == 6)
{
for (i=0; i<10; i++)
{
Wire.beginTransmission(5);
Wire.write(6);
Wire.endTransmission();
```

```
Serial.println(``compensated");
}}
else if (comp == 4)
{
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
Wire.write(4);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if(comp==3)
{
for(i=0;i<100;i++)
{
Wire.beginTransmission(5);
Wire.write(3);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if (comp == 7)
{
for(i=0; i<100;i++)
{
Wire.beginTransmission(5);
Wire.write(7);
Wire.endTransmission();
Serial.println(``compensated");
}}
```

```
else if (comp == 9)
{
for (i=0; i <100; i++)
{
Wire.beginTransmission (5);
Wire.write (9);
Wire.endTransmission ();
Serial.println (``compensated");
}}
else if (comp == 5)
{
for (i=0; i <100; i++)
{
Wire.beginTransmission (5);
Wire.write (5);
Wire.endTransmission ();
Serial.println (``compensated");
}}}
else if (k == 0)
{
Serial.println (``Minor comp");
unsigned long currentMillis = millis ();
if ((unsigned long)(currentMillis - waitMillis)>=
lowCompensationTime)
{
percent=0;
waitMillis = currentMillis;
}
if (comp == 2)
```

```
{
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
Wire.write(2);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if (comp == 8)
{
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
Wire.write(8);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if (comp==6)
{
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
Wire.write(6);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if (comp == 4)
{
for(i=0; i<10;i++)
```

```
{
Wire.beginTransmission(5);
Wire.write(4);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if (comp == 3)
{
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
Wire.write(3);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if (comp == 7)
{
for(i=0; i<10;i++)
{
Wire.beginTransmission(5);
Wire.write(7);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if (comp == 9)
{
for(i=0;i<10;i++)
{
Wire.beginTransmission(5);
```

```
Wire.write(9);
Wire.endTransmission();
Serial.println(``compensated");
}}
else if (comp == 5)
{
for(i=0; i<10; i++)
{
Wire.beginTransmission(5);
Wire.write(5);
Wire.endTransmission();
Serial.println(``compensated");
}}}}}
```

C.4.3 Updated Script for Compensation and Command Forwarding

```
#include <Wire.h>
#include <SoftwareSerial.h>
unsigned long percent;
SoftwareSerial mySerial(10, 11); // RX, TX on the receiver
int compensationTime=300; // C3
int highCompensationTime=700;//C4
int mediumCompensationTime=1000;//C2
int lowCompensationTime=1500;//C1
int turningCompensationTime=300;
int deviationCompensationTime=700;
unsigned long initialMillis=0;
unsigned long got_time;
unsigned long comp;
unsigned long startMillis=0;
unsigned long  waitMillis=0;
```

```
int i;
int n=0;
int k=0;
int m=0;
int t=0;
int s=0;
int u=0;
void setup(void)
{
Serial.begin(57600);
Wire.begin();// set the data rate for the
// SoftwareSerial port
mySerial.begin(4800);
pinMode(13, OUTPUT);// LED
}
void loop(void)
{
while (mySerial.available())
{
unsigned long got_time= mySerial.read();
if (got_time>9){percent=got_time-10;
Serial.println(percent);
t=t+1;}
if(percent==0&&t>=300)
{
if (got_time == 2)
{
Wire.beginTransmission(5);
Wire.write(2);
```

```
Wire.endTransmission();
Serial.println(2);
comp=2;
m=m+1;
}
else if (got_time == 3)
{
Wire.beginTransmission(5);
Wire.write(3);
Wire.endTransmission();
Serial.println(3);
comp=3;
m=1;
}
else if (got_time == 4)
{
Wire.beginTransmission(5);
Wire.write(4);
Wire.endTransmission();
Serial.println(4);
comp=4;
m=0;
}
else if (got_time == 5)
{
Wire.beginTransmission(5);
Wire.write(5);
Wire.endTransmission();
Serial.println(5);
```

```
comp=5;
m=1;
}
else if (got_time == 6)
{
Wire.beginTransmission(5);
Wire.write(6);
Wire.endTransmission();
Serial.println(6);
comp=6;
m=0;
}
else if (got_time == 7)
{
Wire.beginTransmission(5);
Wire.write(7);
Wire.endTransmission();
Serial.println(7);
comp=7;
m=1;
}
else if (got_time == 8)
{
Wire.beginTransmission(5);
Wire.write(8);
Wire.endTransmission();
Serial.println(7);
comp=8;
m=m+1;
```

```
}
else if (got_time == 9)
{
Wire.beginTransmission(5);
Wire.write(9);
Wire.endTransmission();
Serial.println(9);
comp=9;
m=1;
}
else if (got_time == 0)
{
m=0;
n=n+1;
comp=0;
s=0;
Serial.println(k);
}
if(m>5&&m<=10){k=1;}
else if(m>3&&m<=5){k=0;}
else if(m<=3&&m>1){k=10;}
else if(m=0){k=20;}
else if(m=1){k=30;}
}
else if(percent==0&&t<300)
{
if (got_time == 2)
{
if (s<=20)
```

```
{
Serial.println(2);
comp=2;
t=0;
m=m+1;
for(i=0; i<10;i++){
Wire.beginTransmission(5);
Wire.write(2);
Wire.endTransmission();
Serial.println("initial anticipation 2");
n=n+1;
if(n==10){n=20; comp=0;}
}}
else if (s>20)
{
Serial.println(2);
comp=2;
t=0;
m=m+1;
for(i=0; i<50;i++){
Wire.beginTransmission(5);
Wire.write(2);
Wire.endTransmission();
Serial.println(``anticipated 2");
n=n+1;
if(n==50){n=60; comp=0;}
}
}}
else if(got_time == 3)
```

```
{
if (s<=20)
{
Serial.println(3);
comp=3;
t=0;
m=1;
for(i=0; i<10;i++){
Wire.beginTransmission(5);
Wire.write(3);
Wire.endTransmission();
Serial.println(``initial anticipation 3");
n=n+1;
if(n==10){n=20; comp=0;}
}}
else if (s>20)
{
Serial.println(3);
comp=3;
t=0;
m=1;
for(i=0; i<50;i++){
Wire.beginTransmission(5);
Wire.write(3);
Wire.endTransmission();
Serial.println("anticipated 3");
n=n+1;
if(n==50){n=60; comp=0;}
}
```

```
}
}
else if (got_time == 4)
{
if (s<=20)
{
Serial.println(4);
comp=4;
t=0;
 m=0;
for(i=0; i<10;i++){
Wire.beginTransmission(5);
Wire.write(4);
Wire.endTransmission();
Serial.println(``initial anticipation 4");
n=n+1;
if(n==10){n=20; comp=0;}
}}
else if (s>20)
{
Serial.println(4);
comp=4;
t=0;
m=0;
for(i=0; i<50;i++){
Wire.beginTransmission(5);
Wire.write(4);
Wire.endTransmission();
Serial.println("anticipated 4");
```

```
n=n+1;
if(n==50){n=60; comp=0;}
}}}
else if (got_time == 5)
{if (s<=20)
{
Serial.println(5);
comp=5;
t=0;
m=1;
for(i=0; i<10;i++){
Wire.beginTransmission(5);
Wire.write(5);
Wire.endTransmission();
Serial.println(``initial anticipation 5");
n=n+1;
if(n==10){n=20; comp=0;}
}}
else if (s>20)
{
Serial.println(5);
comp=5;
t=0;
m=1;
for(i=0; i<50;i++){
Wire.beginTransmission(5);
Wire.write(5);
Wire.endTransmission();
Serial.println("anticipated 5");
```

```
n=n+1;}}}
else if (got_time == 6)
{
if (s<=20)
{
Serial.println(6);
comp=6;
t=0;
m=0;
for(i=0; i<10;i++){
Wire.beginTransmission(5);
Wire.write(6);
Wire.endTransmission();
Serial.println(``initial anticipation 6");
n=n+1;
if(n==10){n=20; comp=0;}
}}
else if (s>20)
{
Serial.println(6);
comp=6;
t=0;
m=0;
for(i=0; i<50;i++){
Wire.beginTransmission(5);
Wire.write(6);
Wire.endTransmission();
Serial.println(``anticipated 6");
n=n+1;
```

```
if(n==50){n=60; comp=0;}
}}}
else if (got_time == 7)
{
if (s<=20)
{
Serial.println(7);
comp=7;
t=0;
m=1;
for(i=0; i<10;i++){
Wire.beginTransmission(5);
Wire.write(7);
Wire.endTransmission();
Serial.println(``initial anticipation 7");
n=n+1;
if(n==10){n=20; comp=0;}
}}
else if (s>20)
{
Serial.println(7);
comp=7;
t=0;
m=1;
for(i=0; i<50;i++){
Wire.beginTransmission(5);
Wire.write(7);
Wire.endTransmission();
Serial.println(``anticipated 7");
```

```
n=n+1;
if(n==50){n=60; comp=0;}
}}}
else if (got_time == 8)
{
if (s<=20)
{
Serial.println(8);
comp=8;
t=0;
m=m+1;
for(i=0; i<10;i++){
Wire.beginTransmission(5);
Wire.write(8);
Wire.endTransmission();
Serial.println(``initial anticipation 8");
n=n+1;
if(n==10){n=20; comp=0;}
}}
else if (s>20)
{
Serial.println(8);
comp=8;
m=m+1;
for(i=0; i<50;i++){
Wire.beginTransmission(5);
Wire.write(8);
Wire.endTransmission();
Serial.println(``anticipated 8");
```

```
n=n+1;
if(n==50){n=60; comp=0;}
}}}
else if (got_time == 9)
{
if (s<=20)
{
Serial.println(9);
comp=9;
t=0;
m=1;
for(i=0; i<10;i++){
Wire.beginTransmission(5);
Wire.write(9);
Wire.endTransmission();
Serial.println(``initial anticipation 9");
n=n+1;
if(n==10){n=20; comp=0;}
}
else if (s>20)
{
Serial.println(9);
comp=9;
t=0;
m=1;
for(i=0; i<50;i++){
Wire.beginTransmission(5);
Wire.write(9);
Wire.endTransmission();
```

```
Serial.println(``anticipated 9");
n=n+1;
if(n==50){n=60; comp=0;}
}}}
else if (got_time == 0)
{
m=0;
n=n+1;
comp=0;
s=0;
}
if(m>5&&m<=10){k=1;}
else if(m>3&&m<=5){k=0;}
else if(m>10&&m<100000){k=2;}
else if(m<=3&&m>1){k=10;}
else if(m=0){k=20;}
else if(m=1){k=30;}
}
}
if (!mySerial.available()&&percent>0)
{
if (k==0){
Serial.println(``k0");
unsigned long currentMillis = millis();
if ((unsigned long)(currentMillis - initialMillis) >=
mediumCompensationTime){percent=0;
initialMillis = currentMillis;
}
if (comp==2)
```

```
{
for(i=0; i<100;i++){
Wire.beginTransmission(5);
Wire.write(2);
Wire.endTransmission();
Serial.println(``Med compensated 2");
}}
else if (comp==8)
{
for(i=0; i<100;i++){
Wire.beginTransmission(5);
Wire.write(8);
Wire.endTransmission();
Serial.println(``Med compensated 8");
}}
else if(n>0&&comp==0){percent=0;
n=1;
Serial.println(``True");}}
else if (k==1)
{
Serial.println(``k1");
unsigned long currentMillis = millis();
if ((unsigned long)(currentMillis - initialMillis) >=
compensationTime){percent=0;
initialMillis = currentMillis;
}
if (comp==2)
{
for(i=0; i<100;i++){
```

```
Wire.beginTransmission(5);
Wire.write(2);
Wire.endTransmission();
Serial.println(``compensated 2");
}}
else if (comp==8)
{
for(i=0; i<100;i++){
Wire.beginTransmission(5);
Wire.write(8);
Wire.endTransmission();
Serial.println("compensated 8");
}}}
else if (k==2)
{
Serial.println(``k2");
unsigned long currentMillis = millis();
if ((unsigned long)(currentMillis - startMillis) >=
highCompensationTime){percent=0;
startMillis = currentMillis;
}
if (comp==2)
for(i=0; i<100;i++){
Wire.beginTransmission(5);
Wire.write(2);
Wire.endTransmission();
Serial.println(``H compensated 2");
}}
else if (comp==8)
```

```
{
for(i=0; i<100;i++){
Wire.beginTransmission(5);
Wire.write(8);
Wire.endTransmission();
Serial.println(``H compensated 8");
}}
}
else if(k==10)
{
Serial.println(``Low");
unsigned long currentMillis = millis();
if (unsigned long(currentMillis - waitMillis)>=
lowCompensationTime)
{percent=0;
waitMillis = currentMillis;
}
if (comp==2)
{
for(i=0; i<10;i++){
Wire.beginTransmission(5);
Wire.write(2);
Wire.endTransmission();
Serial.println(``L compensated 2");
}}
{
for(i=0; i<10;i++){
Wire.beginTransmission(5);
Wire.write(8);
```

```
Wire.endTransmission();
Serial.println(``L compensated 2");
}}}
else if (k==20)
{
Serial.println(``Turning");
if ((unsigned long)(currentMillis - waitMillis)
>=turningCompensationTime){percent=0;
waitMillis = currentMillis;
}
else if (comp==4)
{
for(i=0; i<10;i++){
Wire.beginTransmission(5);
Wire.write(4);
Wire.endTransmission();
Serial.println(``Turning");
}}
else if (comp==6)
{
for(i=0; i<10;i++){
Wire.beginTransmission(5);
Wire.write(6);
Wire.endTransmission();
Serial.println(``Turning");
}}}
else if (k==30)
{
Serial.println(``Deviation");
```

```
unsigned long currentMillis = millis();
if ((unsigned long)(currentMillis - waitMillis)>=
deviationCompensationTime){percent=0;
waitMillis = currentMillis;
}
else if (comp==5)
{
for(i=0; i<100;i++){
Wire.beginTransmission(5);
Wire.write(5);
Wire.endTransmission();
Serial.println(``deviation");
}}
else if (comp==7)
{
for(i=0; i<100;i++){
Wire.beginTransmission(5);
Wire.write(7);
Wire.endTransmission();
Serial.println(``deviation");
}}
else if (comp==9)
{
for(i=0; i<100;i++){
Wire.beginTransmission(5);
Wire.write(9);
Wire.endTransmission();
Serial.println(``deviation");
}}
```

```
else if (comp==3)
{
for (i=0; i<100; i++){
Wire.beginTransmission(5);
Wire.write(3);
Wire.endTransmission();
Serial.println(``deviation");
}}}}}
```

C.4.5 Command Execution

Command are retrieved form slave B over the TWI interface. This code enables the motor controller board to operate motors with respect to the instruction contain in the received packet. Each command received will be executed instantly. At the end of a packet execution, the controller waits for 50 milliseconds after which it shuts down the motors if no other command is received.

```
#include ``motordriver_4wd.h"
#include <seeed_pwm.h>
#include <Wire.h>
void receiveEvent (int howMany){}
void setup()
{
Wire.begin(5);
Wire.onReceive(receiveEvent);
MOTOR.init(); //all pins initiated
}
void loop()
{
while(Wire.available())
{
unsigned long c = Wire.read();
```

```
if (c == 2)
{
MOTOR.setSpeedDir(30, DIRF);
}
else if (c == 5)
{
MOTOR.setSpeedDir1(15, DIRF);
MOTOR.setSpeedDir2(35, DIRF);
}
else if (c == 8)
{
MOTOR.setSpeedDir(30, DIRR);
}
else if (c == 9)
{
MOTOR.setSpeedDir1(15, DIRR);
MOTOR.setSpeedDir2(35, DIRR);
}
else if (c == 3)
{
MOTOR.setSpeedDir1(35, DIRF);
MOTOR.setSpeedDir2(15, DIRF);
}
else if (c == 7)
{
MOTOR.setSpeedDir1(35, DIRR);
MOTOR.setSpeedDir2(15, DIRR);
}
else if (c == 4)
```

```
{
MOTOR.setSpeedDir1(40, DIRR);
MOTOR.setSpeedDir2(40, DIRF);
}
else if (c == 6)
{
MOTOR.setSpeedDir1(40, DIRF);
MOTOR.setSpeedDir2(40, DIRR);
}
}
while(!Wire.available())
{
delay(50);
MOTOR.setStop1();
MOTOR.setStop2();
}}
```

Appendix D: Proposed Approach Based on Machine Learning

D.4 Concept Review

The present work could have also used the concepts of machine learning to compensate for packet losses. Machine learning is useful in the sense that it allows computers (e.g. CPD) to learn from a previous experience. A number of issues could have been solved today if computers were enabled to do so. These problems do not necessarily relate to data storage or speed. Take for instance, the growing need for flight controls or production processes or any other complex tasks. Each of these tasks would just be a burden on human operators without the contribution from computers. Hence, Machine learning model was thought to remove the limitation to useful applications of computers [1][4].

In order to understand the learning concept of computers, three pillards affiliated to the field of machine learning were highlighted. These are supervised, reinforcement and unsupervised learning.

D.4.1 Supervised Learning

Supervised learning is a broad strategy which enables a computer to do exactly what it is taught. In other words, both inputs and outputs are given to the computer. To explain, a computer uses a prediction function to find out how it should behave in the future based on valid input/output pairs. These pairs are also known as training sets [1]. Prediction is useful when a computer has to deal with an input that was implicitly or explicitly part of those given to it in the training set. For example knowing that input variables (also "called features") λ, Υ and $\propto$ have respective dependent variables ϑ, ξ and η, the supervised learning algorithm should be able to also predict the output for μ which was not in the training set it has. So the purpose of supervised learning is to enable the computer to use a table of training sets, and predict reasonable outcomes.

D.4.2 Reinforcement Learning

It is the second pillar of machine learning where a computer learns from its mistakes. To explain this differently, a computer is not told the right thing to do, but instead it gets a feedback corresponding to its actions. For example, suppose a scenario whereby a computer has to drive safe without having any knowledge on traffic rules. If it succeeds it will then be rewarded. However for each traffic violation it gets a penalty. So a reinforcement learning algorithm will enable the computer to try to drive safe by minimizing penalties and maximizing reward.

D.4.3 Unsupervised Learning

This is the third form of machine learning. The latter does not require a training set unlike its predecessors. It is rather based on observations and the relationships in the universe. In unsupervised learning, all data either have the same label or have no label at all [1]. So a computer has no idea how the data was created and what the rules used to create it were. However, an unsupervised learning algorithm will determine patterns in a given data set and try to make a sense out of it. Next, it may decide to group the data in clusters based on similarities in their characteristics. As a matter of fact, take for instance social networks such as facebook, linkedIn, facetime, etc. The algorithm figures out how people relate to one another and makes suggestions to those with friends in common to also become friends [1][4].

Thesis and Dissertation Services